Politecnico di Torino

Dipartimento di Ingegneria Meccanica e Aerospaziale - DIMEAS
Corso di Laurea Magistrale in Ingegneria Meccanica

Tesi di Laurea Magistrale

Implementazione di pianificatori di percorsi basati su campionamento per la pianificazione del moto di un manipolatore a sette gradi di libertà

Relatore:
Prof. Giuseppe Quaglia

Candidato:
Francesco Cavallo

Correlatore:
Giovanni Colucci

Anno Accademico 2021/2022

A mia madre e a mio padre.

Abstract

Il presente elaborato nasce dall'esigenza emersa dal mondo agricolo di impiegare tecnologie robotiche innovative per supportare le operazioni tradizionali svolte dall'uomo e renderle più efficienti. L'agricoltura di precisione, in questo senso, permette di raccogliere dati, elaborarli e analizzarli al fine di adoperare opportune strategie volte all'ottimizzazione della gestione delle attività agricole nonché all'aumento della qualità dei prodotti e alla massimizzazione della produttività del suolo.

L'obiettivo specifico del presente elaborato è quello di implementare un braccio robotico collaborativo su di una piattaforma mobile dedicata all'agricoltura di precisione. A tale scopo, si è effettuata un'analisi dei pianificatori di traiettorie presenti nella libreria *open-source* OMPL (acronimo di *Open Motion Planning Library*), i quali si basano su metodi probabilistici per la generazione di percorsi privi di collisioni, nel rispetto di eventuali vincoli cinematici imposti dalla struttura del manipolatore.

La base mobile su cui il manipolatore è stato installato è il prototipo "*Agri.q02*": un rover a due moduli e otto ruote motrici dotato di un pannello solare orientabile, progettato presso il dipartimento di Ingegneria Meccanica del Politecnico di Torino per attività di agricoltura di precisione. Tra le sue varie peculiarità, il pannello fotovoltaico del *rover* permette di immagazzinare l'energia solare e consente l'atterraggio di droni volanti.

L'implementazione di un manipolatore seriale su di una piattaforma mobile permette, nell'ambito dell'agricoltura di precisione, di svolgere alcune attività tipiche del mondo dell'agricoltura tradizionale, quali:

- Il campionamento e monitoraggio di terreni;
- La somministrazione mirata e localizzata di fertilizzanti;
- La raccolta a fini di campionamento del raccolto.

Il *cobot* (abbreviazione di *collaborative robot,* ovvero robot collaborativo) impiegato è stato il modello *Jaco2* a sette gradi di libertà prodotto dall'azienda *Kinova*, modellato e comandato in ambiente *ROS (Robotic Operating System)* al fine di svolgere operazioni di raccolta di grappoli d'uva ed oggetti della medesima grandezza.

Lo studio è stato condotto riproducendo l'ambiente di lavoro del manipolatore e testando diversi algoritmi di pianificazione del moto al fine di individuarne il più efficace per l'applicazione specifica. A tal scopo, sono state svolte oltre 250 prove, distribuite su diversi pianificatori *single* e *multi-query*, e sono stati valutati i seguenti parametri:

- Indice di successo di pianificazione;
- Tempo computazionale per la pianificazione del moto.

Per il singolo pianificatore con i migliori parametri di valutazione, inoltre, è stato individuato il dominio, all'interno dell'area di lavoro del manipolatore, nel quale l'indice di successo del processo di pianificazione si mantiene superiore al 90%.

L'elaborato fornirà, inizialmente, un breve cenno sullo stato dell'arte in merito all'agricoltura di precisione. Successivamente, presenterà alcuni cenni relativi agli algoritmi di pianificazione del moto per manipolatori articolati: rispetto a questi ultimi sarà, inoltre, narrato il principio di funzionamento del meta-sistema operativo *ROS,* strumento impiegato per il controllo.

Negli ultimi capitoli, infine, verrà descritta nel dettaglio la metodologia utilizzata per l'esecuzione delle prove che hanno portato a individuare un pianificatore ottimale rispetto a tempo computazionale ed indice di riuscita per la specifica applicazione.

Sommario

Indice figure

Indice tabelle

1. Robotica di servizio

La robotica di servizio è una branca della robotica che comprende tutte quelle applicazioni tecnologiche studiate, progettate e realizzate per aiutare gli esseri umani nello svolgimento di attività giornaliere. Solitamente i robot di servizio si occupano di sostituire l'uomo nello svolgimento di mansioni ripetitive, pericolose, in ambienti ostili oppure a distanza.

L'*ISO* (*International Organization for Standardization* – Organizzazione Internazionale per la Standardizzazione) definisce un "Robot di Servizio" come un robot o tutte quelle apparecchiature in grado di eseguire compiti utili per l'uomo escludendo le applicazioni di automazione industriale: per esempio a questa categoria appartengono i robot che operano nel campo della logistica, in ambito medico, agricolo, nella pulizia professionale di ambienti, nella robotica di ispezione e di manutenzione ([7] - International Federation of Robotics, IFR, s.d.).

> *"La scienza non è nient'altro che una perversione, se non ha come suo fine ultimo il miglioramento delle condizioni dell'umanità"*
>
> ***Nikola Tesla***

1.1. Robotica di servizio per l'agricoltura di precisione

Al giorno d'oggi si sente parlare sempre più spesso di *Smart Farming*: per *Smart Farming* si intende l'impiego di tecnologie avanzate per ottimizzare la gestione e la produttività delle aziende agricole.

La combinazione di nuove tecnologie come l'*IoT* (*Internet of Things*) e il *Cloud Computing* possono favorire questa ottimizzazione di risorse, anche attraverso l'impiego congiunto di robot e di sistemi di *AI* (Intelligenza Artificiale).

Le applicazioni robotiche di supporto agli agricoltori possono offrire un giusto contributo all'incremento della produttività. Il *FAO* (*Food and Agricolture Organization*) stima che entro il 2050 ci sarà un incremento della popolazione mondiale del 34% e pertanto sarà necessario incrementare ed ottimizzare la produzione di generi alimentari.

Tra le mansioni per cui questi sistemi tecnologicamente avanzati e autonomi possono trovare il loro impiego ci sono:

- Aratura dei campi: trattore senza equipaggio teleguidato a distanza;
- Semina;
- Potatura;
- Monitoraggio delle colture (impiego di droni per il controllo della crescita);
- Raccolta di precisione di frutta e verdura;
- Valutazione ed ispezione dei campi;
- Imballaggio.

Di seguito verranno descritti alcuni prototipi di applicazioni robotiche impiegate nell'agricoltura di precisione.

Prototipo *Ladybird UGV system*

Il primo progetto presentato è noto come ***Ladybird UGV System*** ed è stato realizzato nel 2014 dall'*Australian Centre for Field Robotics* (*ACFR*) presso l'università di Sydney. È stato progettato con l'idea di fornire uno strumento flessibile a supporto della ricerca e dello sviluppo di applicazioni robotiche da impiegare nell'industria della produzione di ortaggi commerciali.

Nello specifico l'impiego di questa applicazione robotica ha l'obiettivo di monitorare la crescita delle colture di diverse varietà genetiche in diverse condizioni ambientali in modo da poter estrapolare dati e informazioni importanti, volti a migliorare e ottimizzare la crescita delle colture.

Tipicamente queste informazioni vengono estrapolate da ricercatori direttamente su campo.

Questo sistema è stato progettato al fine di soddisfare i criteri di modularità e flessibilità che sono elementi chiave per la realizzazione di un sistema efficiente, efficace e versatile.

Ladybird UGV System è stato progettato e realizzato con i seguenti componenti principali:

1- Meccanismo di azionamento: quattro motori elettrici identici, modulari. Ciascun motore presenta due assi meccanicamente disaccoppiati, uno per l'orientamento della ruota e uno per azionare la ruota. Il disaccoppiamento riduce al minimo gli sforzi causati dall'irregolarità del suolo e il consumo energetico.
2- Sistema di alimentazione: una batteria a Litio Ferro Fosfato (LiFePO4) si occupa di immagazzinare e fornire energia elettrica. La presenza di pannelli fotovoltaici sui pannelli superiori permette la ricarica della batteria.
3- Calcolatore: è presente un solo computer con processore *Intel Core i7-3610QE* e 16 GB RAM, su cui gira *Ubuntu Linux*.
4- Sensoristica: sono stati installati diverse tipologie di sensori in modo tale da poter rilevare informazioni dai raccolti in autonomia. La presenza di sensori è inoltre importante al fine di evitare ostacoli e rilevare le file delle colture.
5- Manipolatore: il sistema è dotato di un braccio robotico *UR5* a 6 *Degrees of Freedom* (d'ora in poi *DoF*). Il braccio robotico può essere impiegato per la manipolazione (es. aratura) e può occuparsi di spruzzare pesticidi in maniera mirata grazie alla presenza di un ugello sull'organo terminale.
6- Telaio: *Ladybird UGV* è regolabile meccanicamente in larghezza e in altezza, non di minor importanza è la possibilità di regolare l'angolo delle coperture solari. La possibilità di modificare la configurazione del robot permette l'adattamento di quest'ultimo a molteplici tipologie di colture. La regolazione avviene allentando manualmente i bulloni, regolando la configurazione fisica e serrando nuovamente i giunti allentati. È opportuno che questo processo avvenga direttamente su campo.

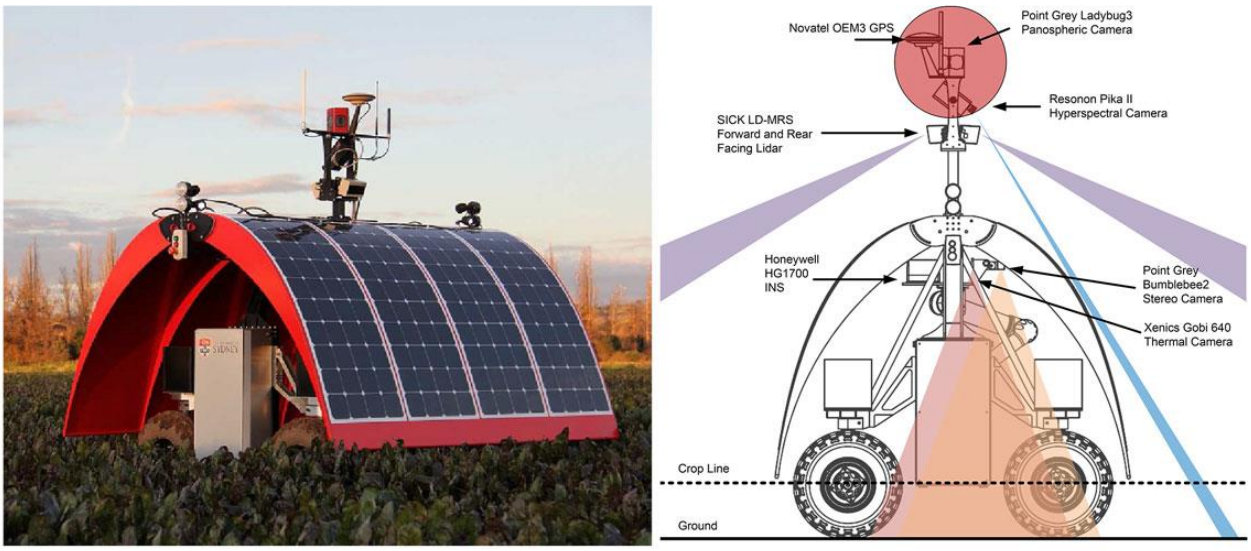


Figura 1 - Rover Ladybird UGV immerso in un campo di ortaggi. Fonte ([1]- James Underwood, 2017)

Figura 2 - Rover Ladybird UGV in un campo. Fonte ([1]- James Underwood, 2017)

Fonte ([1]- James Underwood, 2017).

Prototipo *SWEEPER*

Un'altra soluzione di notevole interesse è il prototipo ***SWEEPER:*** il primo robot al mondo realizzato per la raccolta di peperoni dolci impiegato in una serra commerciale. È stato sviluppato grazie alla partnership della *Wageningen University & Research*, di *De Tuindershoek BV* (coltivatore di peperoni), della *Umea University*, della *Ben-Gurion University*, della *Research Station for Vegetable Cultivation* e della *Bogaerts Greenhouse Logistic*.

È stato progettato per operare in un sistema di coltura commerciale a filare singolo, in una coltivazione con frutti non a grappolo e poca occlusione fogliare. Per questa soluzione si assume che il frutto raccolto non debba essere danneggiato in alcun modo, così come gli altri frutti, le altre piante e i cavi.

Il sistema robotico *SWEEPER* è stato progettato e realizzato con i seguenti componenti principali:

1- Un braccio robotico industriale *Fanuc LR Mate 200iD 6DoF*;
2- L'organo terminale del robot presenta un meccanismo per la potatura, un sistema di imaging e un dispositivo per la raccolta dei frutti;
3- Elevatore a forbice che permette di regolare l'altezza della base del robot;
4- Un computer principale di alta fascia con unità di elaborazione grafica GPU;
5- Controllori a logica programmabile (PLC):
    a. Un controllore Arduino che monitora le operazioni del carrello: movimento lungo la fila e l'elevazione a forbice del carrello;
    b. Un altro PLC si occupa di controllare le funzioni a basso livello dell'organo terminale.
6- Sensori;
7- Compressore per la produzione di aria compressa;
8- Contenitore dove riporre la frutta raccolta;
9- Alimentazione a batterie ricaricabili 24 V;

Figura 3 - Due peperoni rilevati dal sistema di visione integrato sul polso del robot. Fonte ([2]- Boaz Arad, 2020)

Qui di fianco è riportata un'immagine di due peperoni rilevati dal sistema di *imaging* presente sul polso del manipolatore. Si può notare come il sistema di riconoscimento immagine riesca a rilevare i peperoni mostrando la posizione degli oggetti nello spazio tridimensionale riferiti al SR centrato nella camera.

Qui di seguito sulla sinistra è riportata l'immagine raffigurante il robot immerso all'interno di uno dei filari mentre si accinge a potare un peperone.

Sulla destra, invece, possiamo notare il robot non a lavoro con il box contenente l'elettronica di controllo e di potenza. In questa immagine si possono vedere i componenti principali che lo costituiscono.

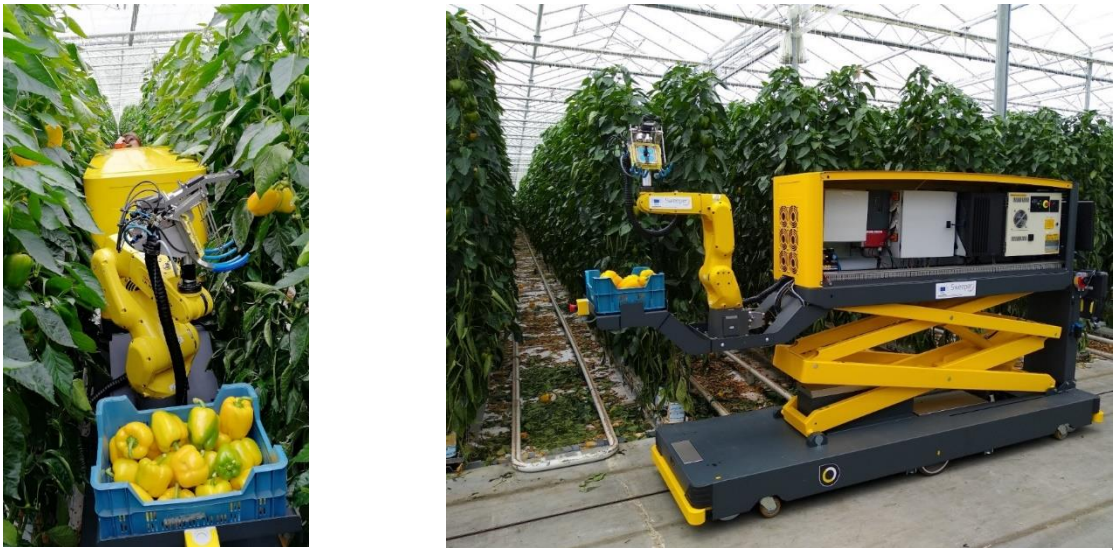


Figura 4 - Robot immerso all'interno dell'ambiente di lavoro per cui è stato realizzato. Fonte ([2]- Boaz Arad, 2020)

Fonte ([2]- Boaz Arad, 2020).

Prototipo *GRAPE*

Un'altra soluzione è nota come **GRAPE:** *Ground Robot for vineyArd Monitoring and ProtEction* ed è un progetto sviluppato dal lavoro congiunto del Politecnico di Milano (POLIMI), il *Technology Center di Barcellona* (*EURECAT*) e l'azienda *VITIROVER*.

L'obiettivo di questa applicazione è di migliorare le operazioni di monitoraggio ed ispezione dei vigneti, i quali, sovente, sono soggetti a malattie di origine fungina e/o a malattie causate dall'attacco di insetti e virus.

Il robot così realizzato ha il compito di installare sulla vite un dispenser di feromone, il cui rilascio è in grado di inibire l'attacco della filossera. La filossera è un insetto che in breve tempo provoca gravi danni alle radici e la conseguente morte della pianta che attacca.

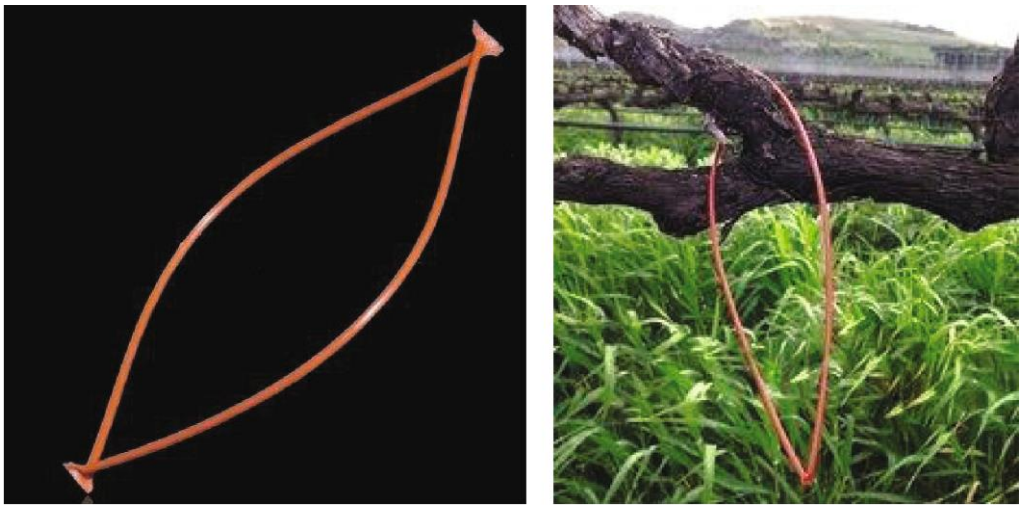


Figura 5 - Sulla sinistra è riportata l'immagine del dispenser di feromone, mentre sulla destra, lo stesso dispenser è stato posizionato correttamente sulla vigna. Fonte ([3]- Ferran Roure, 2017)

Solitamente queste operazioni vengono svolte manualmente in un periodo compreso tra la fine di marzo e gli inizi di aprile, pertanto, le piantagioni sono sprovviste di foglie e la pianta presenta esclusivamente i rami principali dato che i piccoli rami vengono potati.

Il prototipo di seguito è stato pensato e progettato con i seguenti componenti:

1- *HUSKY Robot UGV*: rover terrestre realizzato da *Clearpath Robotics*. Solitamente questo rover viene impiegato in ambiti di ricerca. *HUSKY* è un robot progettato specificamente per soluzioni outdoor e missioni nei campi, essendo in grado di navigare su diverse tipologie di terreno e/o pendenze;

2- Sensoristica:
   - un dispositivo *GPS* e un dispositivo *IMU,* essenziali per una buona localizzazione e navigazione;
   - una stereo camera e una *LIDAR 3D* fondamentali per la ricostruzione approssimativa della vigna. L'impiego combinato di questi strumenti serve per l'ottenimento di una ricostruzione di ciascuna pianta al fine di un corretto posizionamento del dispenser e di un corretto monitoraggio dello stato di salute della pianta;

3- Manipolatore: *KINOVA Jaco2 6DoF* equipaggiato di un laser 2D sul polso del robot al fine di verificare le collisioni on-line.

Come si può notare dall'immagine, il laser 2D presenta una configurazione *eye-in-hand* ovvero con il sensore posizionato in prossimità dell'organo terminale.

Il controllo e la pianificazione del robot sono stati gestiti da un pacchetto *MoveIt* in ambiente *ROS*.

Figura 6 - Prototipo completo pronto per essere portato su campo. Fonte ([3]- Ferran Roure, 2017)

Fonte ([3]- Ferran Roure, 2017).

1.2. Prototipo *Agri.q02*

La stesura di questa tesi nasce dall'esigenza di realizzare un rover teleguidato a distanza da impiegare in vigneti per il monitoraggio delle colture e la raccolta di campioni, quali, terreno, fogliame e frutti, utili alla valutazione dello stato di salute delle colture.

Il prototipo ***Agri.q02***, rientra in quel gruppo di applicazioni dedicate alla robotica di servizio impiegata nell'agricoltura di precisione.

L'idea di base è quella di sviluppare un *UGV* (*Unmanned Ground Vehicle* - Veicolo Terrestre Senza Equipaggio), denominato appunto *Agri.q02*, dedito al monitoraggio delle colture (nello specifico di vigneti) che consente, allo stesso tempo, l'atterraggio, il trasporto e la ricarica di droni volanti d'ora in poi denominati *UAV* (*Unmanned Aerial Vehicle* - Veicolo Aereo Senza Equipaggio).

L'*UAV* e l'*UGV* dotati di opportuni sensori, verranno impiegati per il monitoraggio di vigneti e/o colture in modo da poter adoperare scelte accurate basate su dati concreti.

Agri.q02 è inoltre equipaggiato di un robot collaborativo *Kinova* modello *Jaco2* a 7 gradi di libertà così da poter raccogliere campioni di terreno, foglie o grappoli d'uva.

Il rover *Agri.q02* è stato pensato per il monitoraggio di vigneti che possono trovarsi su un terreno pianeggiante oppure in collina. Si assume, pertanto, la distanza tra due filari successivi, variabile tra 1.5 e 2.5 m.

Tenendo conto dell'ambiente in cui il rover si troverà a lavorare, vengono rispettati i seguenti requisiti:

- Dimensioni: 2 m di lunghezza, 1 m di profondità e 0.7 m di altezza;
- Peso complessivo 118 kg in modo da:
    - essere facilmente trasportabile;
    - ridurre al minimo la compattazione del suolo;
    - ridurre il consumo energetico per il compimento delle operazioni.
- Capacità di salire e scendere pendenze oltre i 20° di inclinazione;
- Mobilità fluida su terreni irregolari e buona manovrabilità;
- Massima velocità in movimento che non dovrà superare 1 m/s;

- Capacità di raggiungere la posizione orizzontale del pannello superiore indipendentemente dall'inclinazione e dalle condizioni del suolo su cui si trova l'*UGV*. Questo permetterà al drone e al braccio robotico di lavorare in maniera indipendente;
- Angolo di beccheggio massimo di 40° per il pannello solare in modo da poter garantire in buona parte delle condizioni una buona esposizione del pannello alla luce del sole;
- Affidabilità e sicurezza dell'intero sistema robotico. I filari e le colture dovranno essere sempre e comunque preservati.

Di seguito verranno esplicati il sistema di locomozione del rover e il meccanismo impiegato per regolare l'orientazione dell'angolo di beccheggio del piano di atterraggio degli *UAV*.

Figura 7 - Agri.q02 immerso in un vigneto. Sul retro è possibile notare il braccio robotico KINOVA Jaco2 a 7 gradi di libertà.

Sistema di locomozione e dispositivo di posizionamento

Il veicolo è realizzato di un modulo anteriore impiegato per la trazione del rover ed un telaio posteriore sul quale sono fissati il pannello solare e il braccio robotico *Kinova Jaco2* a sette gradi di libertà.

Tenendo conto che l'*UGV* dovrà muoversi su un terreno irregolare, il rover è dotato di otto ruote in modo tale da garantire un'ampia superficie di contatto tra veicolo e terreno.

Per la guida del rover sono previste due unità motorizzate indipendenti (L e R – *Left and Right* – Sinistra e Destra).

Ciascuna unità motorizzata è realizzata con una coppia di ruote montate su un bilanciere e collegate al box anteriore nel giunto C_F (C_{FR} a destra e C_{FL} a sinistra).

Entrambe le unità motorizzate anteriori sono dotate di un motore elettrico e riduttore, che, attraverso un collegamento catena-pignone regola la velocità di rotazione delle coppie di ruote sulla destra e sulla sinistra in maniera indipendente in modo da definire la traiettoria del rover.

Il box anteriore contiene e preserva il sistema di alimentazione elettronico del rover e la componentistica per il controllo.

Il modulo anteriore ed il telaio posteriore presentano un giunto di collegamento nel punto Cv indicato nella Figura 8.

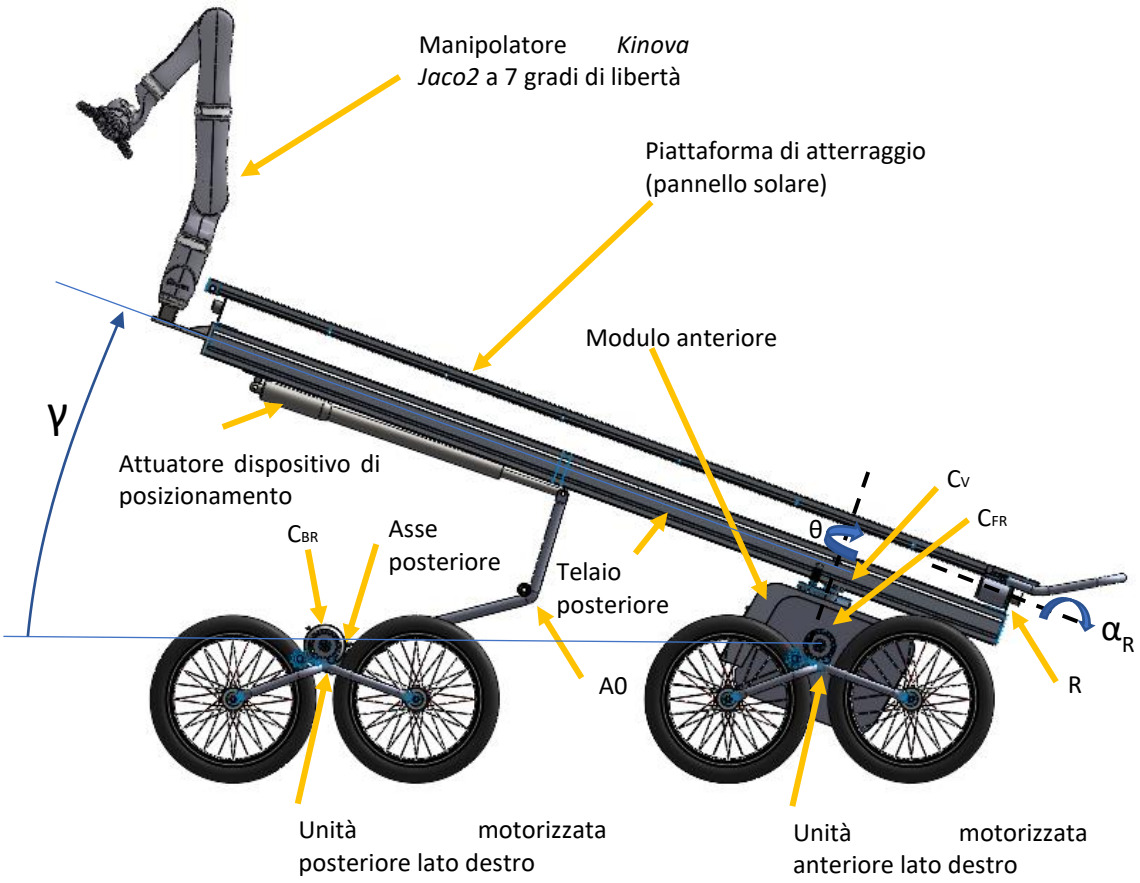


Figura 8 - Componenti principali del prototipo Agri.q02.

Le unità motrici posteriori sono collegate all'asse posteriore nel giunto C_B (C_{BR} a destra e C_{BL} a sinistra) e sono realizzate entrambe con un motore elettrico e un riduttore, che, attraverso un collegamento catena-pignone regola la coppia erogata alle ruote. Questi motori vengono attuati esclusivamente per salire terreni con pendenza particolarmente ripida. In questo caso viene applicato un controllo di coppia, in modo da non interferire sulla traiettoria imposta dalle unità motrici anteriori.

Com'è possibile apprezzare nell'immagine che segue, la rotazione di ciascun bilanciere garantisce un ridotto spostamento dell'intero veicolo nel caso si dovesse verificare uno spostamento verticale di una singola ruota, causato da un eventuale ostacolo presente sul terreno (es. una pietra o un gradino).

Figura 9 – Le due immagini immortalano Agri.q02 in due istanti successivi al fine di mostrare la ridotta entità dello spostamento dell'intero veicolo nel superamento di un ostacolo (nel caso specifico un gradino).

La piattaforma di atterraggio del robot *UAV*, nonché il pannello solare del rover, è supportata dal telaio posteriore. Un motore elettrico controlla l'angolo di rollio del pannello solare attraverso il giunto in R. L'angolo di rollio del pannello solare α_R può variare all'interno di un range compreso tra i $\alpha_R = \pm 20°$.

Il dispositivo di posizionamento controlla l'angolo di beccheggio γ. Il telaio posteriore è collegato al dispositivo di posizionamento attraverso il giunto in A_0 (Figura 8). Il dispositivo di posizionamento è un meccanismo azionato da un motore elettrico per la gestione e il controllo dell'angolo di beccheggio della piattaforma.

L'angolo di beccheggio γ varia all'interno di un range compreso tra $-5° \leq \gamma \leq +40°$.

L'attuazione di questo meccanismo permette di mantenere la piattaforma di atterraggio del drone, orizzontale, in caso di ripide salite. Inoltre, permette sia di orientare il pannello solare in modo tale da poter ricevere la luce solare in maniera adeguata, sia di alzare la base del manipolatore per facilitare la manipolazione e la raccolta di oggetti.

Vengono di seguito riportati alcuni valori caratteristici dell'angolo di beccheggio:

- $\gamma = \gamma_N = 0$, condizioni nominali (N) con il rover *Agriq.02* su una superficie pianeggiante;
- $\gamma = \gamma_d = -5$, movimento di beccheggio della piattaforma di atterraggio per compensare lievi discese (d);
- $\gamma = \gamma_i = +20°$, massima inclinazione della superficie (i) che può essere compensata utilizzando il movimento di beccheggio della piattaforma di atterraggio;
- $\gamma = \gamma_s = +40°$, massima inclinazione del pannello solare (s) per permettere al pannello solare di raccogliere energia solare.

Fonte ([4] - Giuseppe Quaglia, 2019).

1.3. Manipolatore *Kinova Jaco2*

Il braccio robotico che si intende implementare sulla piattaforma *Agri.q02* è un *cobot* (robot collaborativo antropomorfo) prodotto da *Kinova* modello *Jaco2* a 7 gradi di libertà con polso sferico.

Il robot avendo sette gradi di libertà risulta essere una volta ridondante e per questo motivo presenta alcuni vantaggi tra cui:

- La possibilità di muovere il braccio mantenendo inalterata la posa del polso;
- Una maggiore flessibilità nei movimenti;
- Minori difficoltà con le singolarità.

Si intende installare il manipolatore ad una distanza di circa 0.2 m rispetto alla mezzeria del pannello solare e ad una distanza di circa 1.8 m rispetto al giunto di rotazione che permette il beccheggio del pannello stesso. Di seguito un'illustrazione grafica del prototipo:

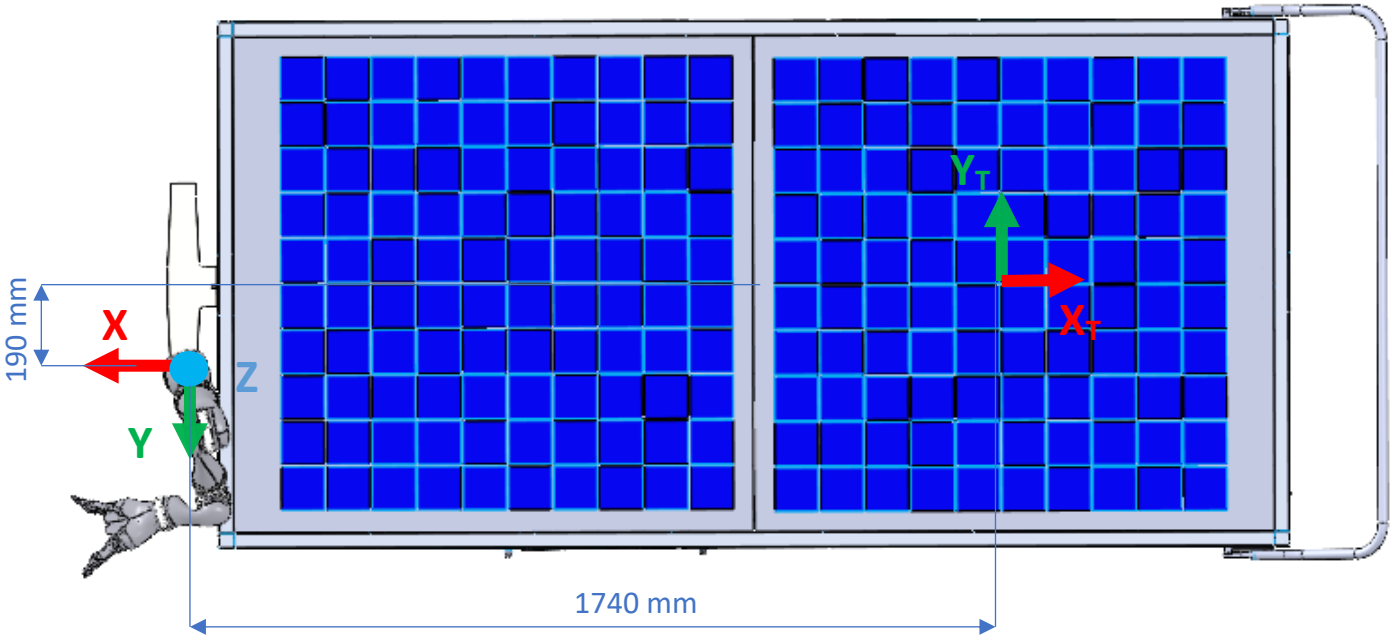


Figura 10 - Posizione della base del manipolator rispetto al SR centrato in T.

Di seguito viene riportata una foto del braccio robotico estrapolata dal manuale d'uso fornito dal costruttore *Kinova*. Si fa presente che l'immagine qui di seguito riportata, illustra un robot con un organo terminale con tre dita anziché due come nel caso oggetto di studio.

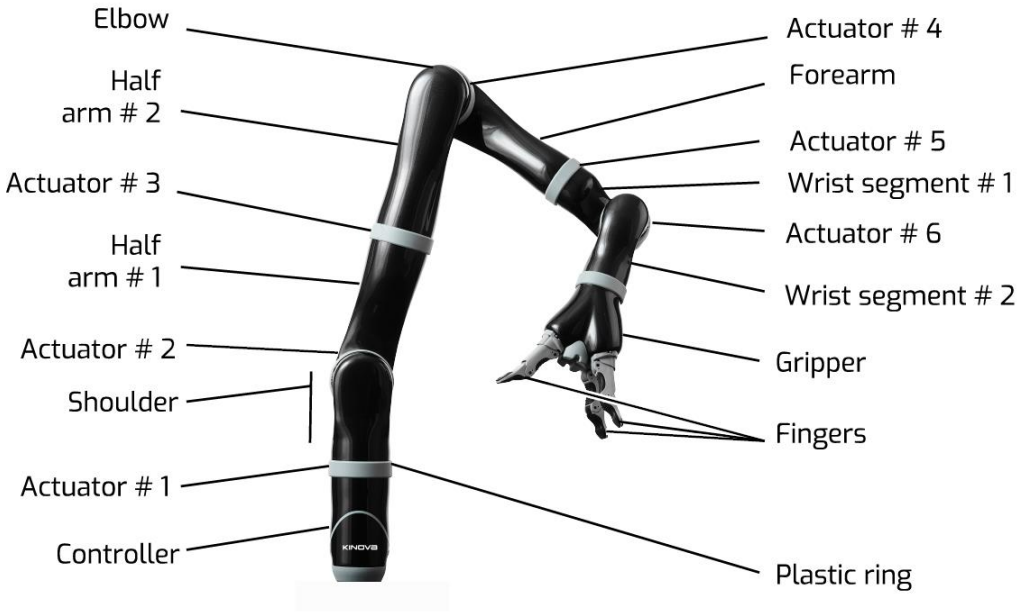


Figura 11 - Braccio robotico collaborativo KINOVA Jaco2 7gdl. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Il controllore del manipolatore si trova alla base dello stesso come è ben illustrato nell'immagine sopra riportata.

Di seguito, invece, è riportata un'immagine raffigurante il pannello situato nella zona posteriore del controller del braccio robotico dove ci sono quattro porte di connessione ed un pulsante di ON/OFF:

- Una porta ethernet per il controllo del braccio e/o la configurazione dello stesso tramite PC;
- Una porta *USB Type-B* per il controllo del braccio e/o la configurazione dello stesso tramite PC;
- Una porta 8-pin per il controllo del robot tramite *Joystick*;
- Una porta 4-pin utilizzata per alimentare il braccio robotico con corrente elettrica.

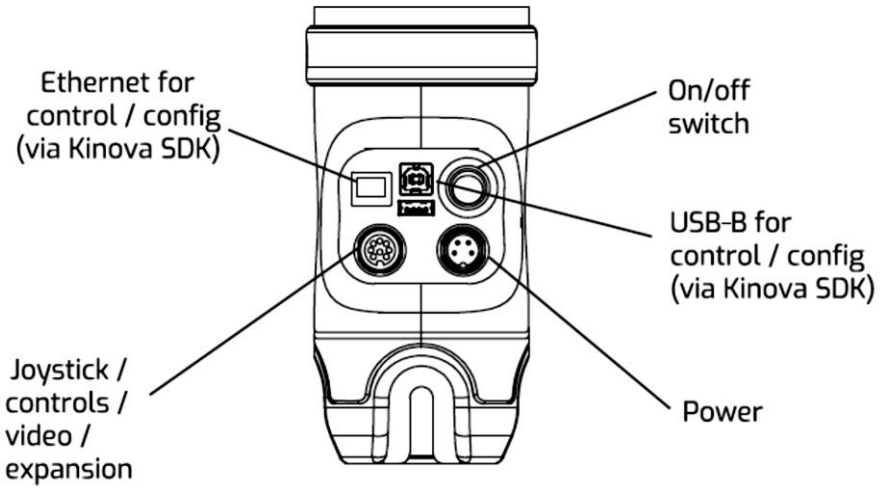


Figura 12 - Parte posteriore del robot dove è possibile inserire i pin necessari a collegare il robot all'alimentazione elettrica, joystick, ethernet e porta USB-B. In alto sulla destra è anche presente un interruttore di accensione e spegnimento. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Le caratteristiche tecniche del robot riportate nel manuale fornito da *Kinova* sono le seguenti:

Caratteristica	Unità di misura	Specifica
Peso totale del robot	[kg]	5,5
Raggio di lavoro	[m]	0,984
Massimo carico a metà raggio di lavoro. Robot sprovvisto di *gripper* e/o pinza.	[kg]	2,4
Massimo carico con braccio in massima estensione. Robot sprovvisto di *gripper* e/o pinza.	[kg]	2,1
Massimo carico a metà raggio di lavoro. Robot provvisto di *gripper* e/o pinza.	[kg]	0.7
Materiale dei link		Fibra di carbonio
Materiale Attuatori		Alluminio
Massima velocità angolare attuatori grandi. Attuatori assi 1-2-3-4. Modello *KINOVA™ Actuator KA-75+.*	[rad/s]	0.63
Massima velocità angolare attuatori piccoli. Attuatori assi 5-6-7. Modello *KINOVA™ Actuator KA-58*.	[rad/s]	0.84
Massima velocità lineare braccio	[m/s]	0,2
Voltaggio alimentazione	[VDC]	18 -> 29
Potenza media	[W]	25 (15 in standby)
Protocollo di Comunicazione		RS485
Resistenza Acqua		IPX2
Temperatura di esercizio	[°C]	-10 -> +40

Tabella 1 - Tabella che riporta le caratteristiche tecniche presenti sul manuale d'uso del robot. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Caratteristiche geometriche del robot e modellazione

Nel capitolo dedicato alla descrizione geometrica del robot, il manuale di uso e manutenzione *Kinova* riporta il braccio completamente esteso. Questa configurazione è riportata nell'immagine sottostante e non è la configurazione di "zero assi" del robot, ma, è la rappresentazione del robot in cui tutti i gradi di libertà sono posti pari a 180°. Si fa presente che per configurazione di ***"zero assi"*** si indica la particolare condizione in cui i giunti del manipolatore hanno valore zero e quindi il vettore $\boldsymbol{q}$ è un vettore nullo $\boldsymbol{q} = [0\ 0\ 0\ 0\ 0\ 0\ 0]$.

Qui di seguito è riportata una tabella contenente le informazioni geometriche relative a ciascun *link*:

Parametro	Descrizione	Lunghezza [m]
D1	Base -> Spalla	0,2755
D2	Prima metà del braccio sopra il giunto di spalla	0,2050
D3	Seconda metà del braccio sopra il giunto di spalla	0,2050
D4	Lunghezza avambraccio (da gomito al polso)	0,2073
D5	Prima lunghezza del polso	0,1038
D6	Seconda lunghezza del polso	0,1038
D7	Lunghezza dal centro del polso al centro dell'organo terminale	0,1600
e2	Spostamento laterale giunto 3-4	0,0133

Tabella 2 - Tabella che riporta le informazioni geometriche di ciascun link. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

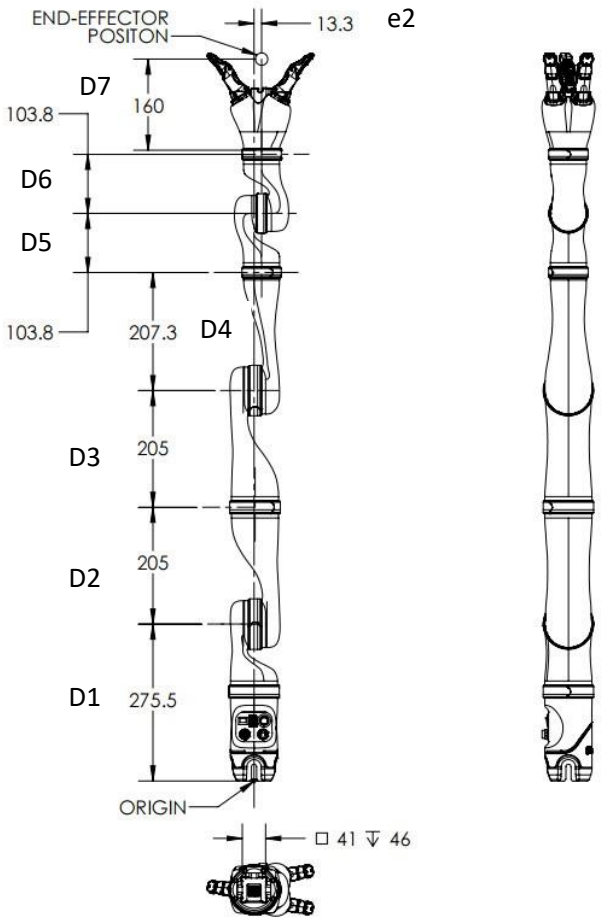


Figura 13 – Robot in totale estensione. Sono riportati anche i parametri geometrici del braccio robotico e le lunghezze di ciascun link. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Si fa presente che a causa dei limiti di giunto dei gradi di libertà (q_2, q_4 e q_6) non è possibile raggiungere la configurazione di ***"zero assi"*** del robot poiché questa configurazione prevede che il braccio sia tutto ripiegato su sé stesso.
Qui di seguito è riportata una tabella contente i limiti di giunto di ciascun grado di libertà:

Giunto	Limite Minimo	Limite Massimo
1	-10000°	10000°
2	47°	313°
3	-10000°	10000°
4	30°	330°
5	-10000°	10000°
6	65°	295°
7	-10000°	10000°

Tabella 3 - In questa tabella sono stati riportati i limiti di giunto del braccio robotico. Dati estrapolati dal manuale d'uso fornito dalla casa produttrice Kinova. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

N.B.: i valori -10000 e +10000 stanno ad indicare assi che non presentano alcun limite di giunto geometrico poiché possono compiere più di una rotazione completa lungo il proprio asse di rotazione.

Di seguito vi è una rappresentazione grafica che riporta il verso di rotazione assunto positivo per ciascun attuatore:

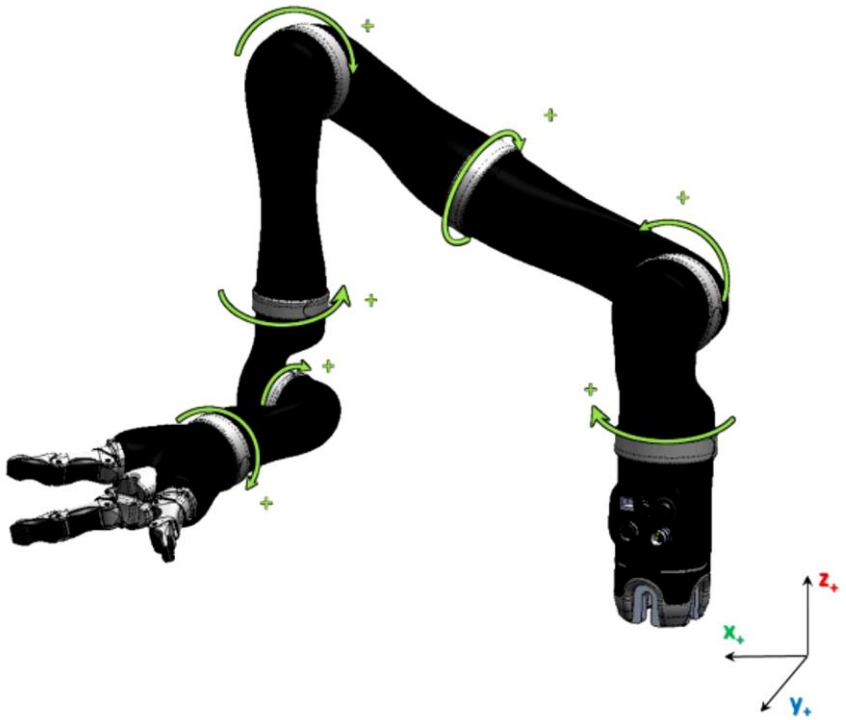


Figura 14 – Braccio robotico su cui sono riportati i versi di rotazione positivi per ciascun giunto. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Nell'immagine che segue vi è la rappresentazione grafica dello spazio operativo del manipolatore:

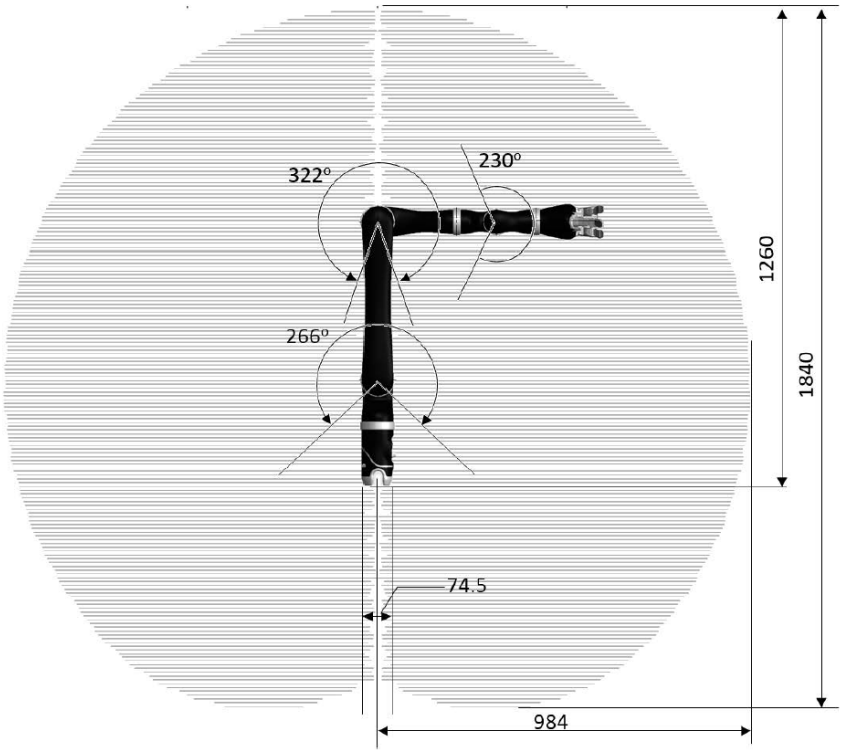


Figura 15 - Spazio operativo del braccio robotico. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Per la scelta dei Sistemi di Rifermento (d'ora in poi S.R.) si è adottata la convenzione di Denavit – Hartenberg (d'ora in poi D.H.). Essa fa sì che una trasformazione geometrica possa essere rappresentata nello spazio euclideo tridimensionale con il numero minimo di parametri (ovvero quattro).

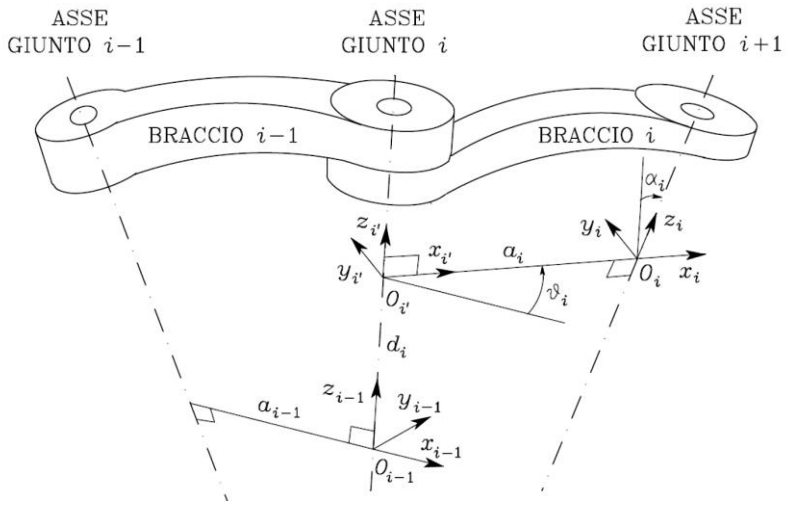


Figura 16 - Parametri di Denavit-Hartenberg per un manipolatore seriale. Fonte ([10] - Robotica. modellistica, pianificazione e controllo, 2008)

- Si sceglie l'asse z_i giacente lungo l'asse del giunto $i+1$;
- Si individua O_i dall'intersezione dell'asse z_i con la normale comune agli assi z_{i-1} e z_i e con O'_i si indica l'intersezione della normale comune con z_{i-1};
- Si assume l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso positivo dal giunto i al giunto $i+1$;
- Si sceglie l'asse y_i in modo da completare una terna levogira.

Definizione non univoca della terna:
- Con riferimento alla terna 0, per la quale la sola direzione dell'asse z_0 risulta specificata: si possono quindi scegliere arbitrariamente O_0 ed x_0;
- Con riferimento alla terna n, per la quale il solo asse x_n risulta soggetto a vincolo (deve essere normale con l'asse z_{n-1}): infatti non vi è giunto $n+1$, per cui non è definito z_n e lo si può scegliere arbitrariamente;
- Quando due assi consecutivi sono paralleli, in quanto la normale comune tra di essi non è univocamente definita;
- Quando due assi consecutivi si intersecano, in quanto il verso di x_i è arbitrario;

I parametri di D.-H. sono:
- a_i : distanza di O_i da O'_i;
- d_i : coordinata di z_{i-1} da O'_i;
- α_i : angolo intorno all'asse x_i tra l'asse z_{i-1} e l'asse z_i valutato positivo in senso antiorario;
- θ_i : angolo intorno all'asse z_{i-1} tra l'asse x_{i-1} e l'asse x_i valutato positivo in senso antiorario;

i	α_i	a_i	d_i	θ_i
1	π /2	0	-D1	q1+pi
2	π /2	0	0	q2
3	π /2	0	-(D2+ D3)	q3
4	π /2	0	-e2	q4
5	π /2	0	-(D4 +D5)	q5
6	π /2	0	0	q6+pi/2
7	π	0	-(D6 + D7)	q7

Tabella 4 - Parametri di D.-H. del robot in esame. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Le matrici impiegate per le trasformazioni di coordinate sono:

Matrice di trasformazione per passare dal SR O_{i-1} al SR $O_{i'}$:

$$\widehat{\boldsymbol{A}}_{\boldsymbol{i'}}^{\boldsymbol{i-1}} = \begin{bmatrix} c\vartheta_i & -s\vartheta_i & 0 & 0 \\ s\vartheta_i & c\vartheta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.1)$$

Matrice di trasformazione per passare dal SR $O_{i'}$ al SR O_i:

$$\widehat{\boldsymbol{A}}_{\boldsymbol{i}}^{\boldsymbol{i'}} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.2)$$

$$\widehat{\boldsymbol{A}}_{\boldsymbol{i}}^{\boldsymbol{i-1}} = \widehat{\boldsymbol{Rot}}(z, \vartheta_i) \cdot \widehat{\boldsymbol{Tras}}(z, d_i) \cdot \widehat{\boldsymbol{Tras}}(x, a_{i-1}) \cdot \widehat{\boldsymbol{Rot}}(x, \alpha_{i-1})) \quad (1.3)$$

Matrice di trasformazione per passare dal SR O_{i-1} al SR O_i:

$$\widehat{\boldsymbol{A}}_{\boldsymbol{i}}^{\boldsymbol{i-1}}(\boldsymbol{q_i}) = \widehat{\boldsymbol{A}}_{\boldsymbol{i'}}^{\boldsymbol{i-1}} \widehat{\boldsymbol{A}}_{\boldsymbol{i}}^{\boldsymbol{i'}} = \begin{bmatrix} c\vartheta_i & -s\vartheta_i c\alpha_i & s\vartheta_i s\alpha_i & a_i c\vartheta_i \\ s\vartheta_i & c\vartheta_i c\alpha_i & -c\vartheta_i s\alpha_i & a_i s\vartheta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.4)$$

La procedura operativa descritta da D.H. per la scelta dei S.R. consiste nell'individuare e numerare consecutivamente gli assi dei giunti ed assegnare le direzioni degli assi $z_0, \dots, z_{n-1}$.

Successivamente, si deve fissare la terna base posizionandone l'origine sull'asse z_0; gli assi x_0 e y_0 sono scelti in maniera tale da ottenere una terna levogira.

Si devono ripetere i seguenti passi per gli assi $i=1, \dots, n-1$:

1. Individuare l'origine O_i all'intersezione di z_i con la normale comune agli assi z_{i-1} e z_i. Se gli assi z_{i-1} e z_i sono paralleli e il giunto i è rotoidale, posizionare O_i in modo da annullare d_i;
2. Fissare l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso positivo dal giunto i al giunto $i+1$;
3. Fissare l'asse y_i in modo da ottenere una terna levogira.

Infine, si deve fissare la terna n, allineando z_n lungo la direzione di z_{n-1} se il giunto n è rotoidale, ovvero scegliendo z_n in maniera arbitraria.

A questo punto si deve costruire per $i=1, \dots, n$ la tabella dei parametri α_i, a_i, d_i e θ_i.

È possibile a questo punto calcolare sulla base dei parametri α_i, a_i, d_i e θ_i le matrici di trasformazione omogenee $\boldsymbol{A_i^{i-1}(q_i)}$ per $i=1, \dots, n$;

Si possono così calcolare $\boldsymbol{T_n^0(q) = A_1^0 \dots A_n^{n-1}}$ che fornisce posizione e orientamento della terna *n* rispetto alla terna 0.

Infine, assegnate $\boldsymbol{T_0^b}$ e $\boldsymbol{T_e^n}$, calcolare la funzione cinematica diretta $\boldsymbol{T_e^b(q) = T_0^b T_n^0 T_e^n}$ che fornisce posizione ed orientamento della terna utensile rispetto alla terna base.

Il manuale di uso e manutenzione del robot riporta la seguente immagine in cui sono stati riportati i SR per ciascun giunto partendo dal SR Base fino ad arrivare al SR centrato nell'organo terminale:

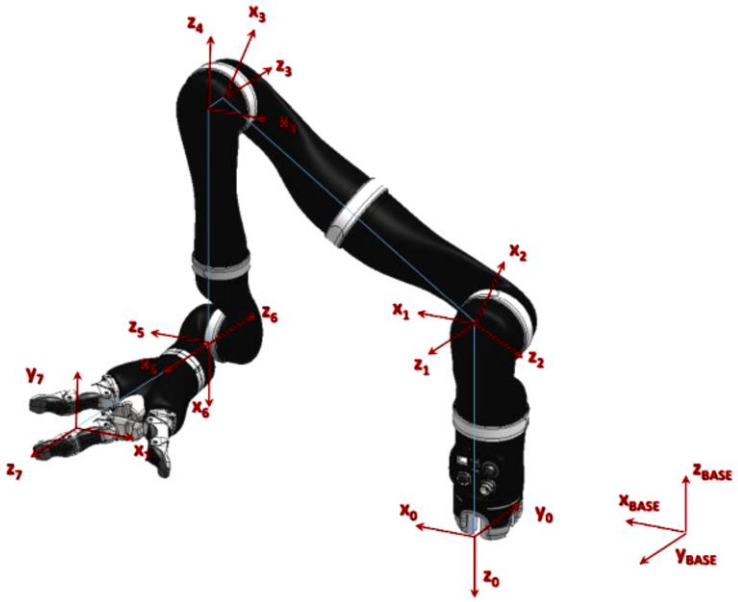


Figura 17 - S.R. del robot per ciascun giunto. Il robot è in configurazione $\bar{q} = [180, 240, 0, 60, 90, 90, 180]$.
In basso sulla destra è riportato il S.R. base da cui partire per eseguire correttamente le trasformazioni di coordinate e conoscere la corretta posizione dell'organo terminale una volta note le coordinate giunto del robot. Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF)

Dato un vettore contenente le informazioni di ciascun giunto $\bar{q} \in \mathbb{R}^{7x1}$:

$$\bar{q} = (q_1, q_2, q_3, q_4, q_5, q_6, q_7)^T \qquad (1.5)$$

È possibile applicare la cinematica diretta per conoscere la posizione dell'organo terminale in funzione della configurazione in spazio giunti applicando la seguente formula:

$$\widehat{\boldsymbol{A}}_{\boldsymbol{Tool}}^{\boldsymbol{Base}} = \widehat{\boldsymbol{A}}_{\boldsymbol{0}}^{\boldsymbol{Base}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{2}}^{\boldsymbol{1}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{3}}^{\boldsymbol{2}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{4}}^{\boldsymbol{3}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{5}}^{\boldsymbol{4}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{6}}^{\boldsymbol{5}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{7}}^{\boldsymbol{6}} \cdot \widehat{\boldsymbol{A}}_{\boldsymbol{Tool}}^{\boldsymbol{7}} \qquad (1.6)$$

Il prodotto delle matrici omogenee è una matrice $\widehat{\boldsymbol{A}}_{\boldsymbol{Tool}}^{\boldsymbol{Base}} \in \mathbb{R}^{4x4}$.

$$\widehat{\boldsymbol{A}}_{\boldsymbol{Tool}}^{\boldsymbol{Base}} = \begin{bmatrix} \boldsymbol{A}_{\boldsymbol{Tool}}^{\boldsymbol{Base}} & \boldsymbol{p}_{\boldsymbol{Tool}}^{\boldsymbol{Base}} \\ 0 & 1 \end{bmatrix} \qquad (1.7)$$

$\boldsymbol{A}_{\boldsymbol{Tool}}^{\boldsymbol{Base}} \in \mathbb{R}^{3x3}$

$$\boldsymbol{p_{Tool}^{Base}} = [x_{Tool}\ y_{Tool}\ z_{Tool}\ ]^T$$

Il vettore p_{Tool}^{Base} conterrà la posizione del Tool rispetto al SR Base.

Si rimanda al capitolo **Appendice - A** per la verifica delle corrette relazioni qui sopra riportate tramite script *Matlab*.

Controllo del robot

Per il controllo del braccio robotico oggetto di studio sono disponibili tre opzioni:

- Controllo tramite *Kinova Joystick*;
- Software di controllo forniti da *Kinova*:
  - *Development Center*;
  - *TorqueConsole*;
- *API* (*Application Programming Interface*) *Control*: *Kinova* mette a disposizione per gli utenti, librerie in linguaggio *C++* per controllare i propri robot.

Controllo del robot tramite Kinova Joystick

Il *Joystick* non è stato uno strumento impiegato per lo sviluppo di questo lavoro di tesi, perciò, si rimanda al manuale d'uso del robot per ulteriori approfondimenti (*Gen2 7DoF User Guide* - https://www.kinovarobotics.com/en/resources/gen2-technical-resources).

Software di controllo forniti da Kinova

Questi software permettono il controllo del robot ad alto livello e non sono stati strumenti utilizzati per lo sviluppo e la stesura di questo lavoro, pertanto, si rimanda al manuale d'uso del robot per ulteriori approfondimenti *(Gen2 7DoF User Guide* - https://www.kinovarobotics.com/en/resources/gen2-technical-resources).

Controllo del robot tramite API

L'*API* (insieme dei file formato *.dll* e *.h*) è scaricabile dal sito web come parte del *Kinova SDK* (*Software Development Kit*). Il *Kinova API* è supportato sia dal sistema operativo Windows che da Ubuntu. *Kinova* offre agli sviluppatori la possibilità di controllare il robot attraverso l'interfaccia *ROS*. Per ulteriori approfondimenti si rimanda alla pagina *Kinova ROS Github*: https://github.com/Kinovarobotics/kinova-ros .

Fonte ([5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF) in bibliografia.

2. *ROS (Robot Operating System)*

L'interfaccia ***ROS*** (acronimo di ***R**obot **O**perating **S**ystem*) è un *framework* che comprende un insieme di librerie di codice e di strumenti atti a semplificare lo sviluppo di sistemi o applicazioni robotiche. Esso fornisce l'astrazione dell'hardware, driver dei dispositivi, librerie, strumenti di visualizzazione, comunicazione a scambio di messaggi tra processi (*message passing*), gestione dei pacchetti, ecc … *ROS* è rilasciato sotto una licenza open source, *BSD*.

ROS è un sistema multi-processo: più programmi girano contemporaneamente in modo parallelo anche su macchine differenti, infatti, permette di far comunicare tutti questi dispositivi e/o processi.

L'ambiente di sviluppo è principalmente basato su due linguaggi di programmazione: *C++* e *Python*. Tuttavia, ci sono dei metodi che permettono di utilizzare altri linguaggi di programmazione coma *Java, JavaScript, Matlab*, ecc…

La parte di comunicazione, che è uno dei core di *ROS*, si basa su un sistema multi-macchina in cui ci sono più processi che girano e si scambiano informazioni contemporaneamente.

ROS è organizzato in pacchetti (***packages***), che costituiscono le applicazioni. Ciascun pacchetto contiene più nodi (***nodes***).

Il singolo processo in *ROS* viene chiamato **nodo**. Un **nodo** non è altro che un programma o un processo in esecuzione, che si occupa di eseguire un'operazione specifica. Un nodo può controllare i *driver* oppure effettuare calcoli.

2.1. Cenni sul funzionamento di *ROS*

Come abbiamo precedentemente accennato, un'applicazione robotica è un insieme di nodi che comunicano e scambiano informazioni all'interno di una rete multi-macchina. Tutto questo viene chiamato ***ROS Graph*** i cui componenti principali sono:

- ***ROS Node*** che possono essere:
  - Processi in esecuzione nella specifica applicazione robotica *ROS*;
  - Codici che girano in quel momento;
  - Nodi appartenenti ad un determinato pacchetto.
- ***ROS Master**:* un nodo speciale che abilita la creazione della rete *ROS* e la parte di comunicazione. Questo nodo viene richiamato lanciando il `roscore`;
- ***ROS Param Server***: un nodo che archivia e modifica i dati accessibili a tutti i nodi. Questo nodo viene richiamato lanciando il `rosparam`;
- ***ROS Message***: un modo per definire in *ROS* come sono incapsulate le informazioni che vengono scambiate all'interno della rete *ROS*. *ROS* implementa tre metodi di comunicazione:
  1) ***Topic**;*
  2) ***Service;***
  3) ***Action.***

I ***Topic*** sono i canali di comunicazione più importanti all'interno di una rete *ROS*. Sono un metodo di comunicazione unidirezionale tra due nodi. Il nodo che invia dati su un *Topic* è chiamato ***Publisher***. Il nodo che riceve dati da un *Topic* è chiamato ***Subscriber***.

Ciascun *Topic* è identificato da una stringa unica (*name*) all'interno della rete.

Su ciascun *Topic*, i dati possono essere pubblicati in un formato specifico (***Message Type***). Il *Message Type* descrive il formato del messaggio. Ogni *Topic* ha un *type* specifico.

I ***Service*** sono canali di comunicazione simili ai *Topic*. Ma a differenza dei *Topic s*ono canali di comunicazione "sincroni": vuol dire che il nodo che manda un messaggio resta in attesa di una risposta. Un *service* è composto da una coppia di messaggi: ***Service Request*** e ***Service Response***. Il codice si blocca nel momento in cui un *server* non riceve una risposta.

Il protocollo di comunicazione è *client/server*:

- Un nodo che invia un messaggio (o una richiesta) su un service è chiamato ***Client***;
- Un nodo che riceve il messaggio (o lo attende) su un service è chiamato ***Server***;

Da riga di comando bisogna digitare:

- `rosservice list // rosservice call/….`

Le ***Action*** sono canali di comunicazione simili ai *Topic* e ai *Services*. Le *Action* però sono pensate per scambiare dati di comunicazione su task complessi o lunghi. Sono canali di comunicazione bidirezionali al fine di scambiare messaggi di ***goal/result/feedback*** tra un nodo *server* ed un nodo *client*. Un nodo *server* propone un'azione che può essere richiamata da un nodo *client* attraverso un ***goal message***. Durante l'esecuzione, il nodo *server* invia periodicamente ***feedback message*** al *client*. Al termine dell'esecuzione, il server invia un messaggio ***result message*** al *client*.

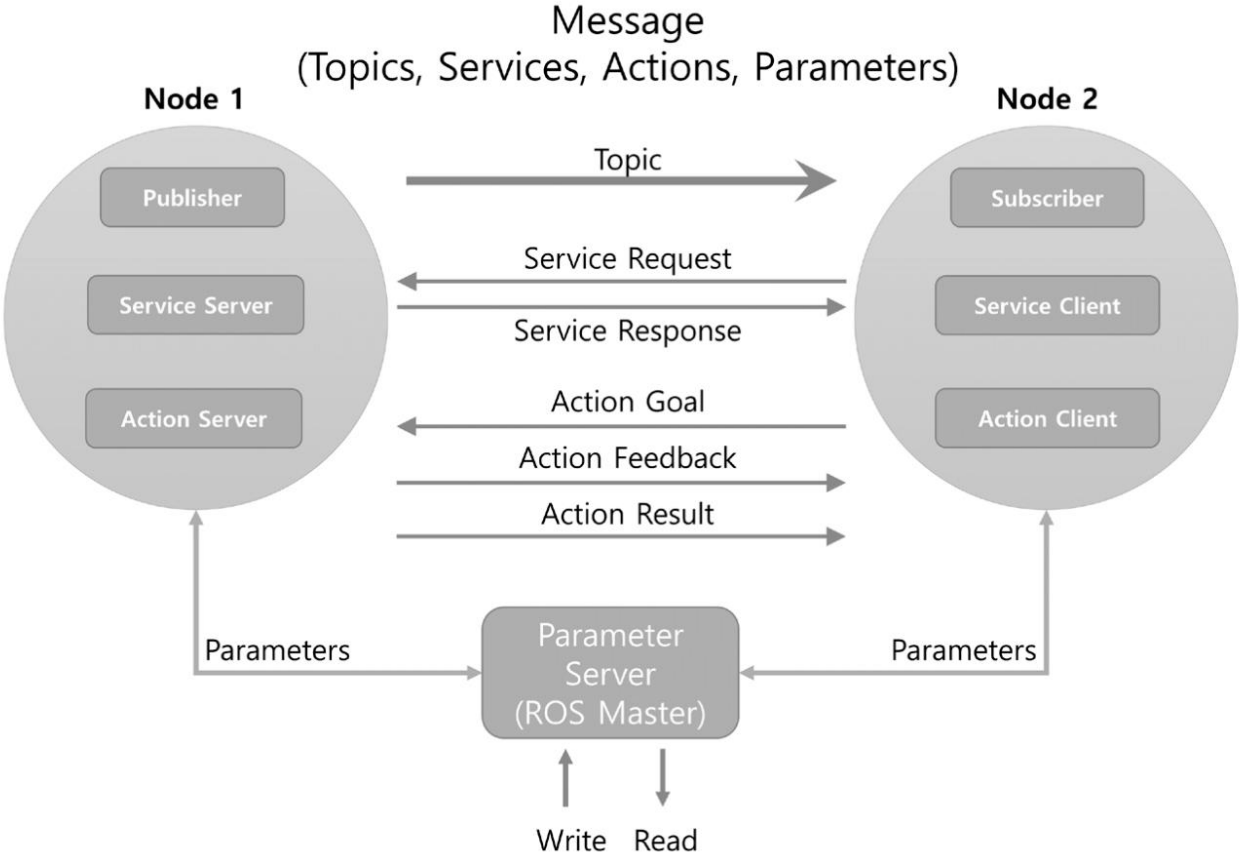


Figura 18 – Schema del funzionamento logico delle comunicazioni all'interno di una rete ROS. Fonte ([6] - Y. Pyo, 2017)

All'interno di *ROS* ci sono una serie di *Tools* che permettono l'analisi, il *debug* e la visualizzazione di quello che sta avvenendo in un'applicazione robotica. Tra i *Tools* più importanti troviamo:

- ***Command Line Tools***;
- ***RVIZ****: ROS Visualizator*;
- ***RQT****;*

- ***ROS Bag**:* serve a registrare le informazioni che vengono scambiate all'interno di un'applicazione *ROS* che possono essere ottimizzate in secondo luogo in fase di *debug*;

Command Line Tools

Una volta aperta la finestra di comunicazione, ci sono una serie di comandi da terminale che permettono di interagire con *ROS*. Tra i comandi principali ci sono:

- ***catkin:** Compilation suite* e creazione di nuovi pacchetti.
  - Tra i comandi principali ci sono `catkin_make`, `catkin_make_install`, `catkin_create_package`;
- ***roscore**:* uno dei principali comandi di *ROS*. Con questo comando viene estratto il master (un nodo speciale in *ROS* che si occupa di creare la rete *ROS* e di mettere in comunicazione tutti i nodi);
- ***rosrun/roslaunch:*** sono due comandi che permettono di lanciare i vari pacchetti *ROS*.
  - ***rosrun***, esegue un singolo programma. Per lanciare un nodo di questo tipo, bisogna mandare da riga di comando:

```
rosrun nome_del_pacchetto_da_lanciare nome_del_nodo_da_lanciare
```

N.B.: all'interno di uno stesso pacchetto possono essere eseguiti più nodi.

  - ***roslaunch***: permette di lanciare una serie di nodi *ROS* tramite un file che viene chiamato proprio *`.launch`* file. Per lanciare un nodo di questo tipo, bisogna mandare da riga di comando:

```
roslaunch nome_del_pacchetto_da_lanciare nome_del_file_da_lanciare.launch.
```

- ***rostopic/rosnode:*** sono comandi che permettono di interagire con l'ambiente *ROS*:
  - ***rostopic*** è un comando *ROS* che permette di interagire con i topic utilizzando il terminale attraverso specifici comandi come ad esempio:
    - `rostopic list` vediamo tutti i *Topic* attivi all'interno della rete *ROS*;
    - `rostopic echo <topic_name>` con questo comando si chiede di pubblicare cosa sta succedendo all'interno di un *Topic*;
    - `rostopic pub <topic_name> <message_type> <message>` con questo comando si inviano (pubblicano) dati o informazioni all'interno di un *Topic* specifico;
    - `rostopic info /<topic_name>` con questo comando si interroga il *Topic*;
  - ***rosnode*** è un comando *ROS* che permette di interagire con i nodi utilizzando il terminale attraverso specifici comandi come ad esempio:
    - `rosnode list` permette di visualizzare la lista di tutti i nodi che sono attivi;
    - `rosnode info /<node_name>` con questo comando si interroga il nodo;
- ***rosmsg/rossrv**;*
- ***rosparam**;*
- ***rosmessage/rosservice**.*
- ecc…

Fonte ([6] - Y. Pyo, 2017).

2.2. MoveIt

MoveIt è un software molto diffuso per la manipolazione dei robot. Esso incorpora i più avanzati sistemi di pianificazione del moto, manipolazione, percezione 3D, cinematica, controllo e navigazione. È rilasciato sotto licenza *BSD* ed è gratuito per utilizzi industriali, commerciali e di ricerca.

Di seguito è riportata un'immagine raffigurante l'architettura di sistema di alto livello per il principale nodo fornito da *MoveIt*, anche noto come `move_group`. Questo nodo funge da integratore, ovvero riunisce tutti i singoli componenti per fornire una serie di *Action* e *Service ROS* che gli utenti possono utilizzare per il controllo delle applicazioni robotiche.

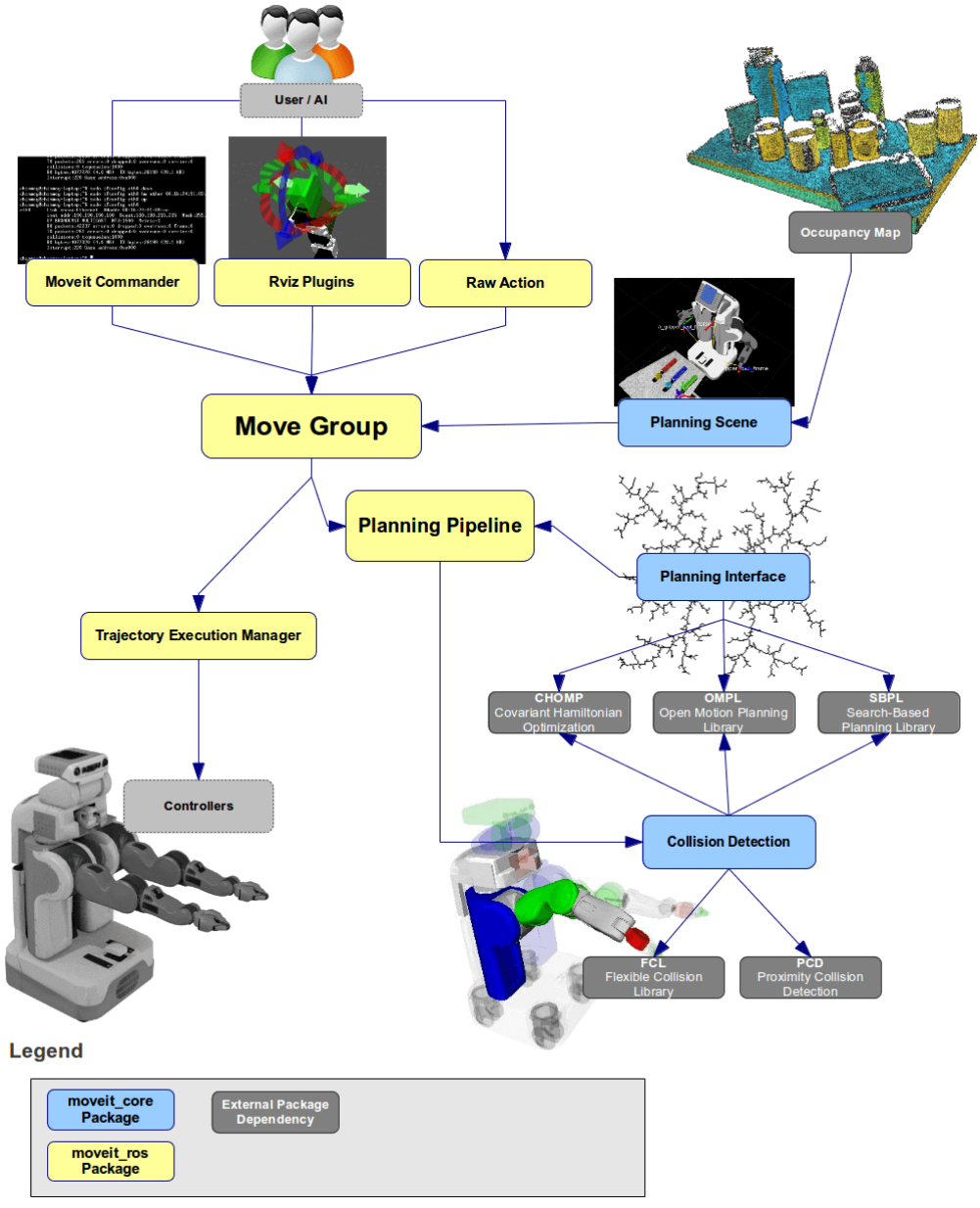


Figura 19 –Architettura di sistema del nodo move_group. Fonte ([8] - MoveIt!, s.d.)

Interfaccia Utente

È possibile accedere alle *Action* e ai *Service* forniti dal nodo `move_group` attraverso tre vie:

- ***C++:*** utilizzando il pacchetto `move_group_interface` che fornisce un'interfaccia semplice da configurare in linguaggio *C++;*
- ***Python***: utilizzando il pacchetto `moveit_commander`;

- ***GUI (Graphical User Interface):*** utilizzando *il Motion Planning plugin di Rviz* (Il visualizzatore *ROS*);

Configurazione

Il nodo `move_group` è un nodo *ROS* che attraverso il ***ROS parameter server*** preleva tre tipi di informazioni:

1- *URDF* – il nodo `move_group` ricerca il parametro `robot_description` *sul ROS parameter server* per ottenere il file *URDF* del robot;
2- *SRDF* – il nodo `move_group` cerca il parametro `robot_description_semantic` *sul ROS parameter server* per ottenere il file *SRDF* del robot. L'*SRDF* viene generato (una volta) dall'utente attraverso il *MoveIt Setup Assistant*;
3- *MoveIt Config* – `move_group` cerca sul *ROS parameter server* altre configurazioni specifiche di *MoveIt* tra cui:
   a. Limiti di giunto;
   b. Limiti cinematici;
   c. Pianificatori del moto;
   d. Ecc… .

   I file di configurazione (*config files*) vengono generati da *MoveIt Setup Assistant* e vengono salvati nella *config directory* corrispondente al pacchetto *MoveIt* del robot.

URDF (Universal Robot Description Format)

Il file *URDF* del robot (*Universal Robot Description Format*) è un file *XML* (*eXtensible Markup Language*) utilizzato in *ROS* per descrivere la proprietà cinematiche, inerziali e geometriche dei link dei robot. Un file *URDF* descrive i giunti e i link di un robot:

- **Giunti.** I giunti connettono due link: un link genitore (*parent*) e un link figlio (*child*). Ciascun giunto è identificato univocamente da un nome. Alcune tipologie di giunto sono:
  - *prismatic*: giunto che permette la traslazione relativa tra il giunto e il link. Dovranno essere specificati gli intervalli e quindi i limiti di giunto;
  - *revolute:* giunto in cui è prevista una rotazione attorno al suo asse. Dovranno essere specificati gli intervalli e quindi i limiti di giunto;
  - *continuous:* giunto simile a revolute, ma che non presenta alcun limite di giunto nella rotazione;
  - *fixed:* giunto fisso, non presenta alcun grado di libertà;
  - *floating:* il giunto può variare di posizione in tutti e sei i gradi di libertà;
  - *planar:* il giunto permette il movimento in un piano perpendicolare all'asse.

  All'interno del tag `<joint>`, devono essere inseriti alcuni parametri tra cui:
    - `<origin>` per specificare la posizione del SR del link figlio rispetto al SR del link genitore. Il giunto sarà posizionato nell'origine del SR del link figlio;
    - `<parent>` per indicare il nome del link genitore;
    - `<child>` per indicare il nome del link figlio;
    - `<axis>` per specificare l'asse del giunto. L'asse viene specificato attraverso un vettore normalizzato (x,y,z) riferito al SR del link figlio. L'asse sarà di rotazione se il giunto è di tipo *revolute/continuous*, di traslazione se il giunto è di tipo *prismatic;*
    - `<limit>` per specificare i limiti di ciascun giunto. L'intervallo viene indicato dichiarando il limite inferiore `lower` e superiore `upper`.

- `effort` permette di definire la massima coppia/forza applicabile al giunto;
- `velocity` permette di definire la massima velocità applicabile al giunto. In caso di giunto *revolute/continuous/prismatic.*

Di seguito sono riportate le righe di codice che descrivono il giunto 1 per la generazione del file *URDF* del manipolatore oggetto di studio:

```
<xacro:macro name="j2s7s300" params="base_parent prefix:=j2s7s300">

<xacro:kinova_armjoint
        joint_name="${prefix}_joint_1" type="continuous"
        joint_origin_xyz="0 0 1"
        joint_origin_rpy="0 3.1416 0"
        parent="${prefix}_link_base"
        child="${prefix}_link_1"
        joint_lower_limit="-2*3.14/180"
        joint_lower_limit="2*3.14/180"
        joint_velocity_limit="${36*J_PI/180}"
        joint_torque_limit="40" />
```

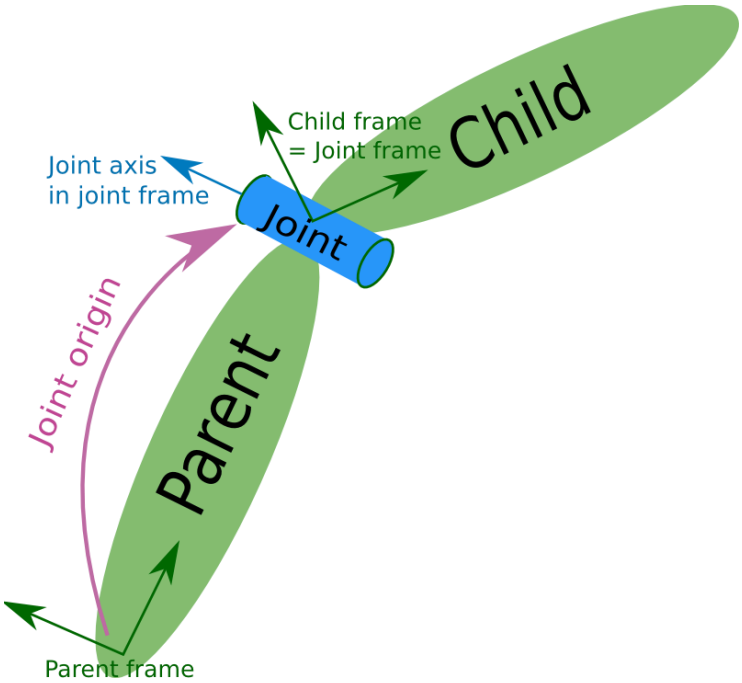


Figura 20 – Esempio di due link connessi da un giunto. Fonte immagine ([23] - XML - Joint, s.d.)

- **Link**. Mentre i giunti descrivono la cinematica del robot, i link ne definiscono le proprietà inerziali. Gli elementi di un link includono la sua massa, un SR origine definisce la posizione e l'orientazione di un SR centrato nel centro di massa e di conseguenza vi è la definizione una matrice di inerzia centrata nel centro di massa del link.

  All'interno del tag `<link>` sono inseriti i seguenti parametri:

  - `<inertial>` questo tag permette di inserire le caratteristiche inerziali del link, e quindi la sua massa e la sua inerzia. La massa è espressa come un numero scalare, mentre l'inerzia è esplicitata da una matrice simmetrica 3 x 3. Il tag `<origin>` permette di specificare la pozione dell'oggetto grafico rispetto al SR del link stesso;

- `<visual>` questo tag permette di inserire informazioni grafiche relative al link. Il link può essere rappresentato da una primitiva grafica (es. cilindro, parallelepipedo o sfera) o oppure da una mesh in formato *.dae* o *.stl.* Il tag `<origin>` permette di specificare la pozione dell'oggetto grafico rispetto al SR del link stesso;
- `<collision>` questo tag permette di esprimere il modello del link specificandolo all'interno del tag `<geometry>`. All'interno del tag `<geometry>` devono essere presenti la geometria e le dimensioni del link oppure deve essere indicata la mesh. Il tag `<origin>` permette di specificare la pozione dell'oggetto grafico rispetto al SR del link stesso.

Di seguito sono riportate le righe di codice che richiamano la mesh relativa al link 1 per la generazione del file *URDF* del robot oggetto di studio:

```
        <xacro:kinova_armlink
                link_name="${prefix}_link_1"
                link_mesh="shoulder"
                mesh_no="1" />
```

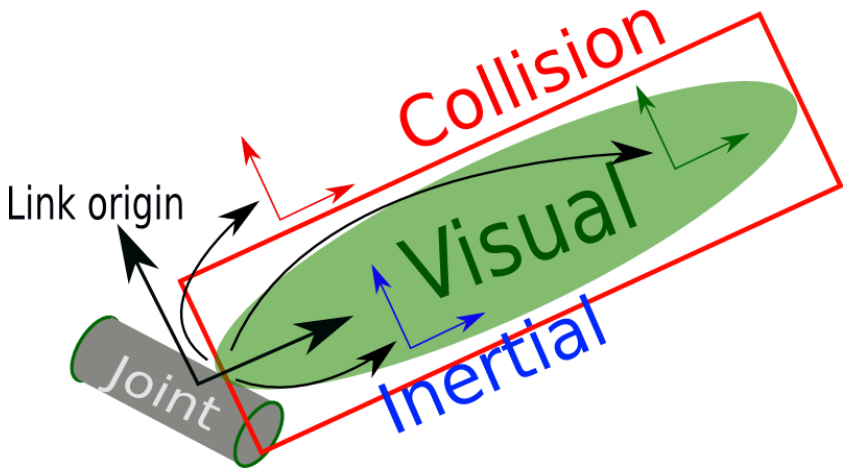


Figura 21 - Esempio di un link e di un giunto. Fonte immagine ([20] - XML - Link, s.d.)

Interfaccia robot

Come già accennato precedentemente, il nodo `move_group`, comunica con il robot attraverso i *ROS topics* e le *Actions*. Esso comunica con il robot per ottenere informazioni sullo stato del robot in tempo reale attraverso:

- *Joint State Information* (Informazioni sullo stato dei giunti);
- *Transform Information* (Informazioni delle trasformazioni);
- *Controller Interface* (Controllori di interfaccia);
- *Planning Scene* (Scena di pianificazione);

Joint State Information

`move_group` si sottoscrive al topic `/j2s7s300_driver/out/joint_state` per determinare le informazioni sullo stato del robot, ovvero per identificare la configurazione dei singoli giunti.

Di seguito un esempio di dati estrapolati dal topic `/j2s7s300_driver/out/joint_state`:

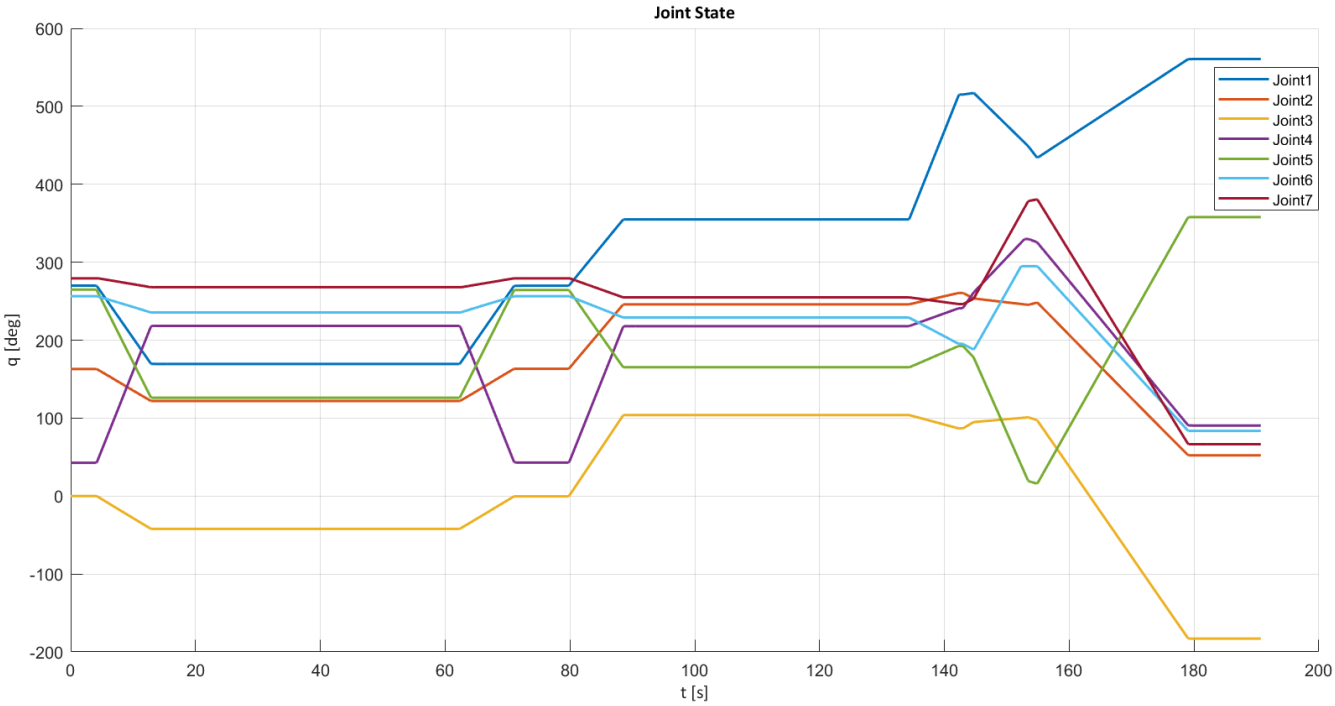


Figura 22 - Grafico in cui sono riportate le configurazioni dei giunti nei vari istanti di tempo nell'esecuzione di generiche movimentazioni. Braccio robotico Kinova Jaco2 a 7 gradi di libertà.

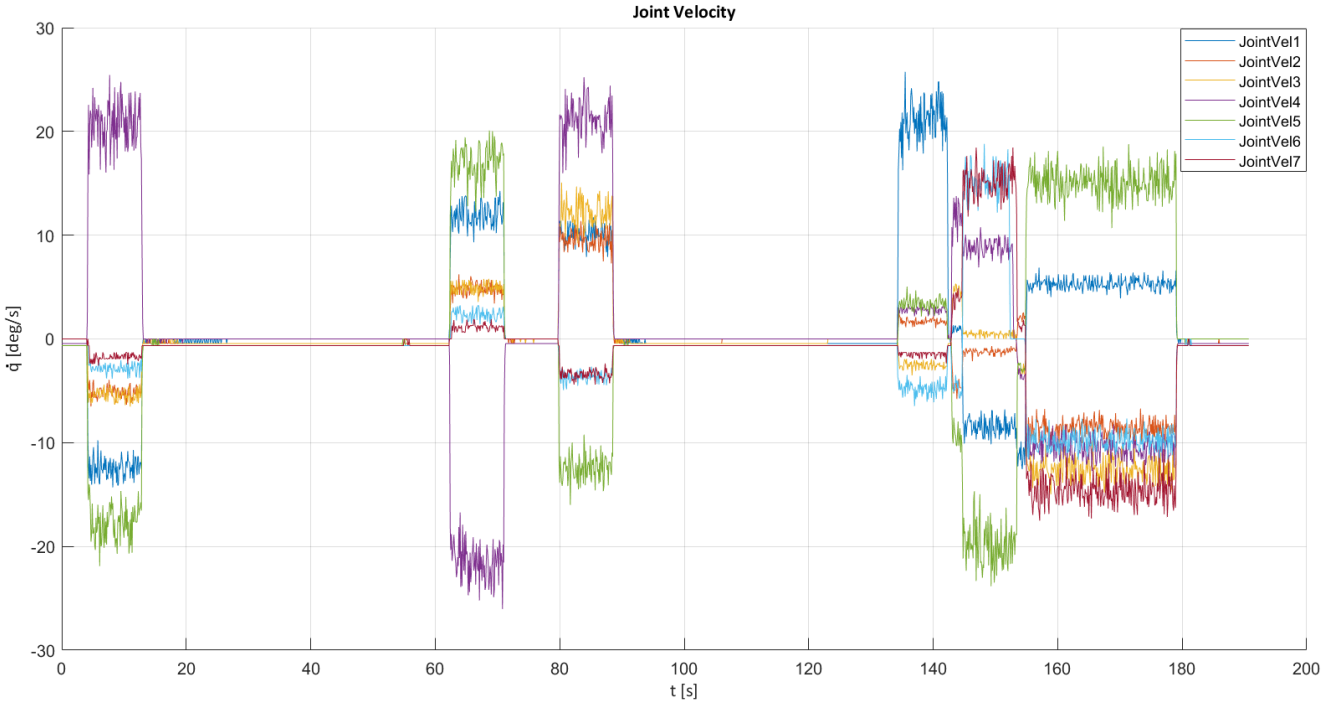


Figura 23 - Grafico in cui sono riportate le velocità angolari dei giunti nei vari istanti di tempo nell'esecuzione di generiche movimentazioni. Braccio robotico Kinova Jaco2 a 7 gradi di libertà.

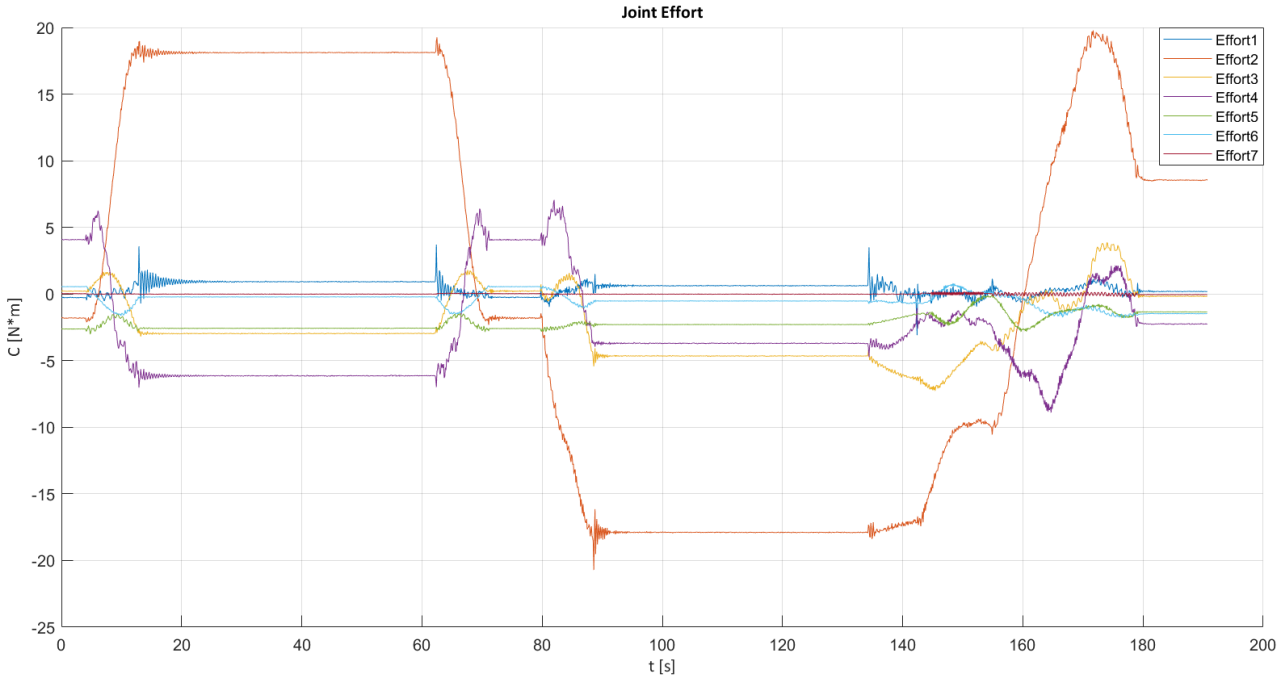


Figura 24 - Grafico in cui sono riportate le coppie erogate dai motori dei giunti nei vari istanti di tempo nell'esecuzione di generiche movimentazioni. Braccio robotico Kinova Jaco2 a 7 gradi di libertà.

Per produrre i grafici qui sopra riportati si sono registrate le informazioni pubblicate sul topic `/j2s7s300_driver/out/joint_state` attraverso il comando:

```
rosbag record -o prova.bag /j2s7s300_driver/out/joint_state
```

Andando a elaborare i dati registrati sul software *MATLAB*, si può notare come, per ciascun giunto, sono disponibili i dati di posizione ('*Position*') espressi in rad (successivamente convertiti in gradi), le velocità ('*Velocity*') dei singoli giunti espresse in rad/s (successivamente convertiti in Deg/s), ed infinite le coppie ('*Effort*') espresse in N*m.

Field	Value
MessageType	'sensor_msgs/JointState'
Header	*1x1 struct*
Name	*13x1 cell*
Position	*13x1 double*
Velocity	*13x1 double*
Effort	*13x1 double*

Figura 25 - Tipologia di informazioni che vengono pubblicate sul nodo joint_state. Per ogni istante di tempo (ogni 0.10000 s) in cui avviene il campionamento di queste informazioni vengono registrate le informazioni di stato, velocità e coppia.

Fonte ([8] - MoveIt!, s.d.).

3. Pianificazione del moto

"Pianificare un moto significa decidere quale movimento deve effettuare il robot per eseguire un compito di trasferimento da una postura iniziale ad una finale senza collidere con ostacoli". Fonte cap.12 ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

Sia B un manipolatore seriale immerso in uno spazio di lavoro $\boldsymbol{W} = \boldsymbol{R^N}$ con N = 2 o 3.
Si considerino $O_1, \dots, O_p$ gli ostacoli, ovvero corpi rigidi presenti nello spazio di lavoro $\boldsymbol{W}$.
Assumendo note le geometrie di $B, O_1, \dots, O_p$ e la posa di $O_1, \dots, O_p$ in W, il problema di pianificazione del moto può essere definito come:
"Data una postura iniziale e una finale di B in $\boldsymbol{W}$, generare, se esiste, un cammino, cioè una sequenza continua di posture, che porti il robot dall'una all'altra e che sia privo di collisioni (inclusi i contatti) tra B e gli ostacoli $O_1, \dots, O_p$; segnalare un fallimento se tale cammino non esiste." Fonte cap.12 ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

Spazio delle configurazioni e distanza

Si definisce con C l'insieme delle configurazioni che il robot può assumere.
Si definiscono le "immagini" o le proiezioni degli ostacoli all'interno dello spazio delle configurazioni CO_i come segue:

$$CO_i = \{\bar{q} \in C : B(\bar{q}) \cap O_i \neq 0\} \qquad (3.1)$$

$$CO = \cup_{i=1}^{p} CO_i \qquad (3.2)$$

$$C_{free} = C - CO = \left\{\bar{q} \in C : B(\bar{q}) \cap \left(\cup_{i=1}^{p} CO_i\right) \neq 0\right\} \qquad (3.3)$$

N.B.: I limiti di giunto vengono considerati come ostacoli all'interno dello spazio C.

La distanza tra due punti $\bar{q}_A$ e $\bar{q}_B$ all'interno dello spazio delle configurazioni è definita dalla norma Euclidea in tale spazio:

$$d(\bar{q}_A, \bar{q}_B) = \left|\left|\bar{q}_A - \bar{q}_B\right|\right| = \sqrt{(q_{1,A} - q_{1,B})^2 + \cdots + (q_{N,A} - q_{N,B})^2} \qquad (3.4)$$

Generazione di traiettorie

"Pianificare un moto privo di collisioni per un robot consiste nel generare un cammino valido e privo di collisioni tra $\bar{q}_{start}$ e $\bar{q}_{goal}$ se queste configurazioni appartengono entrambe alla stessa componente connessa di C_{free}, e di restituire un fallimento in caso contrario". Fonte cap.12 ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

Risulta necessario distinguere a questo punto il significato di percorso e traiettoria:

- ***Percorso***: sequenza di configurazioni (dello spazio operativo o dello spazio giunti) che il manipolatore segue nell'esecuzione del moto assegnato tra $\bar{q}_{start}$ e $\bar{q}_{goal}$. Informazione puramente geometrica.

$s \in [0,1]$

$Percorso \rightarrow \theta(s)$
$s \rightarrow \bar{q}(s)$

$$\theta: [0,1] \rightarrow C$$

- ***Traiettoria:*** percorso per cui è definita una legge temporale. Vengono quindi definiti valori di velocità e/o accelerazione in ogni punto del percorso.

$t \in [0, t_{end}]$

$Traiettoria \rightarrow \Pi$
$t \rightarrow \bar{q}(s(t))$

$$\prod: [0, t_{end}] \rightarrow C$$

Solitamente la procedura di pianificazione del moto prevede che la generazione di un percorso e della rispettiva traiettoria avvengano in momenti differenti. In primo luogo, è prevista la generazione del percorso e in seconda battuta il percorso viene trasformato in traiettoria attraverso l'ausilio di algoritmi di parametrizzazione nel tempo.

Data l'elevata complessità computazionale necessaria per la definizione di un percorso e/o di una traiettoria, non è previsto che questi vengano implementati da un utente in modo dettagliato e per questo motivo vengono impiegati algoritmi appositamente strutturati.

Qualsiasi algoritmo di pianificazione di una traiettoria necessita di alcuni parametri in input come ad esempio:

- il percorso;
- eventuali vincoli sul percorso;
- vincoli dinamici del manipolatore;
- tempo complessivo di esecuzione della traiettoria;
- vincoli sulle massime velocità e accelerazioni dei singoli giunti;
- ecc… .

Definiti i parametri necessari, l'algoritmo di pianificazione della traiettoria, restituisce una sequenza di configurazioni in termini di posizione ed orientamento che l'organo terminale deve assumere in un determinato intervallo di tempo in modo che questo rispetti i vincoli imposti.

Una volta generata la traiettoria è previsto che quest'ultima fornisca i riferimenti al sistema di controllo del manipolatore in modo che venga garantita l'esecuzione della traiettoria rispettando i vincoli di posizione, velocità e accelerazione imposti ai singoli giunti.

La generazione di un percorso (e quindi di una traiettoria) può avvenire sia in spazio giunti che in spazio operativo. Di seguito verranno spiegate in dettaglio le differenze:

- *Spazio operativo:* spazio in cui il robot è immerso e dovrà muoversi. In questo spazio viene individuata la posizione e l'orientamento del manipolatore. Nel caso oggetto di studio lo spazio operativo è lo spazio cartesiano tridimensionale in cui deve anche essere specificato l'orientamento dell'organo terminale. In caso di pianificazione in spazio operativo, a valle della generazione di una traiettoria è necessario richiamare un risolutore di cinematica

inversa per ogni punto della traiettoria così da fornire le opportune coordinate ai giunti per seguire il percorso programmato.

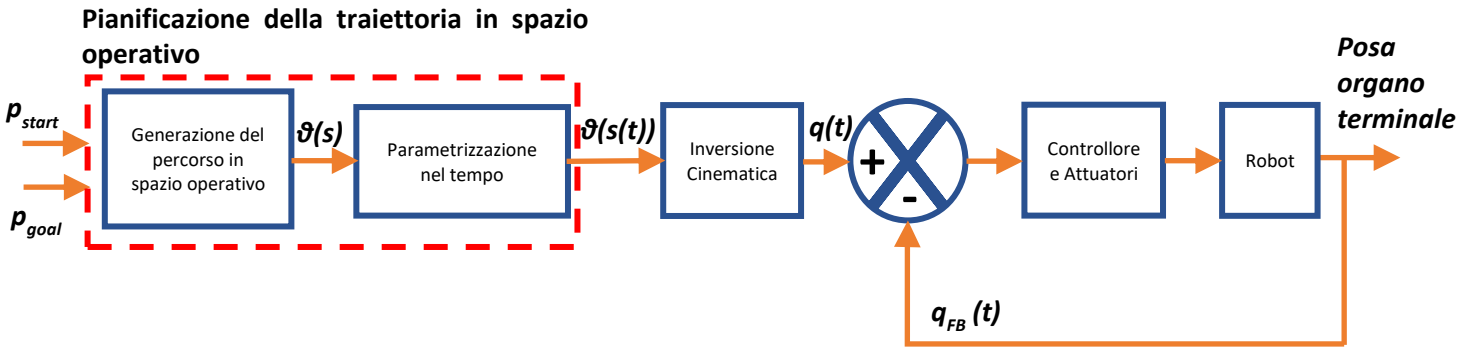


Figura 26 - Diagramma a blocchi che illustra in maniera schematica e sintetica le macro-fasi per la generazione di una traiettoria in spazio operativo

- *Spazio giunti:* spazio vettoriale in cui sono definite le coordinate giunto. Nel caso oggetto di studio $\bar{q} \in \mathbb{R}^{7x1}$. In caso di pianificazione in spazio giunti il risolutore di cinematica inversa viene richiamato una sola volta. Questo avviene nel momento in cui, fornita una posa goal dell'organo terminale, si applica la cinematica inversa per conoscere le coordinate giunto $\bar{q}_{goal}$ da fornire all'algoritmo di pianificazione del moto.

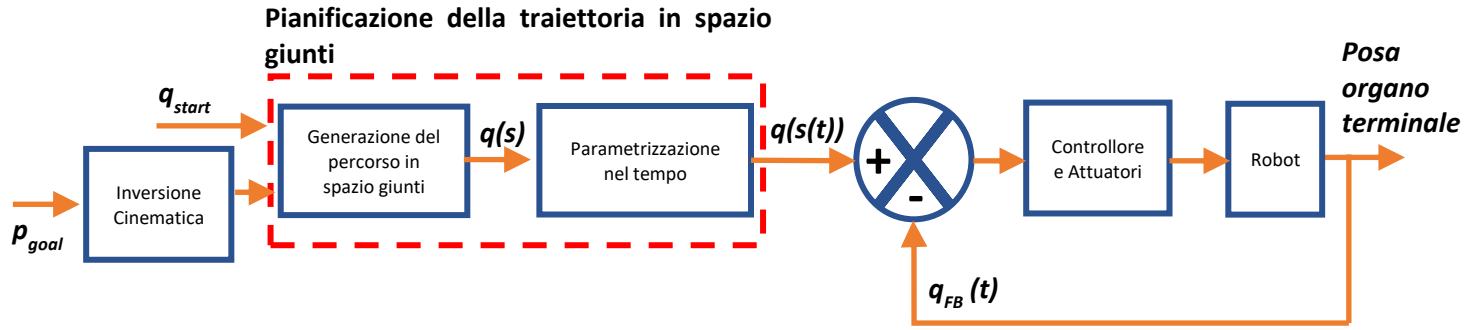


Figura 27 - Diagramma a blocchi che illustra in maniera schematica e sintetica le macro-fasi per la generazione di una traiettoria in spazio giunti

Di seguito verranno esposte alcune delle possibili tecniche adottate per la generazione dei percorsi e successivamente verranno affrontate le tecniche relative alla parametrizzazione nel tempo e quindi alla definizione delle traiettorie all'interno di *MoveIt*.

Assumiamo che d'ora in poi la pianificazione del moto avvenga in spazio giunti.

3.1. Pianificazione del percorso probabilistica basata su campionamento

Un metodo ampiamente utilizzato per la risoluzione di problemi di pianificazione del moto in spazi con elevato numero di dimensioni o per sistemi dotati di dinamica complessa è la pianificazione del moto basata sul campionamento.

I pianificatori che sfruttano questo metodo sono basati sul campionamento, per cui il principio di funzionamento consiste nella ricerca di un certo numero di configurazioni appartenenti a C_{free} che ne rappresenti la connettività in modo adeguato. Tali configurazioni vengono impiegate per la generazione di un percorso utile finalizzato alla soluzione del problema di pianificazione.

I pianificatori basati sul campionamento si definiscono "completi in probabilità", nel senso che la probabilità di trovare una soluzione al problema di pianificazione, se questa esiste, tende ad 1 al tendere all'infinito del tempo di pianificazione.

In funzione delle tecniche adottate, i pianificatori di percorsi si distinguono in due macrocategorie:

- ***Multi-query*.** Al pianificatore viene chiesto di risolvere problemi di pianificazione del moto in un ambiente statico e strutturato e per questo motivo può essere utile la generazione di un C_{free} accurato. Questa struttura può essere utilizzata per risolvere più richieste di pianificazione del moto. Questi pianificatori, infatti, permettono di salvare la mappa delle configurazioni in modo da poterla richiamare in un secondo momento per una nuova pianificazione. Algoritmi di pianificazione del moto di questo tipo sono ***PRM*** e ***PRMstar***;

- ***Single-query*.** Sono pianificatori di percorsi molto diffusi per la risoluzione rapida di problemi di pianificazione. Ogni volta che viene richiamato, il pianificatore, risolve un nuovo problema di pianificazione. A differenza delle tecniche multi-query, questi metodi non prevedono la generazione di una mappa che rappresenti in modo esauriente l'intero spazio libero delle configurazioni C_{free} ma tendono a generare strutture ad albero all'interno dello spazio delle configurazioni ammissibili. Esistono diverse varietà di pianificatori ad albero e per questo lavoro sono stati analizzati i soli pianificatori ***RRT***, ***RRTConnect*** e ***RRT**** (*RRT-star*).

Fonte cap.12 ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

Metodo *PRM (Probabilistic RoadMap)*

Il metodo ***PRM*** (acronimo di ***P**robabilistic **R**oad**M**ap*) utilizza il campionamento dello spazio C_{free} per generare una *RoadMap*, ovvero, un grafo non orientato nello spazio delle configurazioni.

Il metodo iterativo *PRM* prevede prima di tutto la generazione di una configurazione casuale $\boldsymbol{q_{rand}}$ nello spazio delle configurazioni. $\boldsymbol{q_{rand}}$, quindi, viene sottoposta ad una verifica di collisione, attraverso l'utilizzo di relazioni cinematiche e geometriche. Vengono quindi verificate le corrispondenti posizioni nello spazio di lavoro dei corpi che costituiscono il robot richiamando specifici algoritmi in grado di riconoscere eventuali collisioni tra i link del robot e gli ostacoli che lo circondano.

Se $\boldsymbol{q_{rand}}$ non causa collisioni, viene aggiunta alla *RoadMap* esistente. Questa configurazione si cerca di collegarla con cammini elementari ammissibili alle configurazioni "vicine" nella pre-esistente *RoadMap*. Solitamente si usa una definizione di vicinanza basata sulla norma euclidea in C.

La generazione dei cammini elementari tra $\boldsymbol{q_{rand}}$ e una configurazione vicina $\boldsymbol{q_{near}}$ già in *RoadMap* è affidata ad un pianificatore locale.

Al raggiungimento di un numero massimo di iterazioni la procedura di generazione della RoadMap viene arrestata.

Una volta terminata la fase di generazione della *RoadMap*, si verifica se è possibile trovare una soluzione al problema assegnato, connettendo la configurazione iniziale $\boldsymbol{q_{start}}$ e la configurazione goal $\boldsymbol{q_{goal}}$ attraverso cammini elementari ammissibili.

Se non viene trovata una soluzione allo specifico problema di pianificazione, si cerca di arricchire la *RoadMap* eseguendo altre iterazioni. La ricerca del percorso migliore per raggiungere il

nodo $\boldsymbol{q_{goal}}$ partendo dal nodo $\boldsymbol{q_{start}}$ è affidata ad un algoritmo di ricerca su grafi anche noto come *A-star*.

Fonte ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

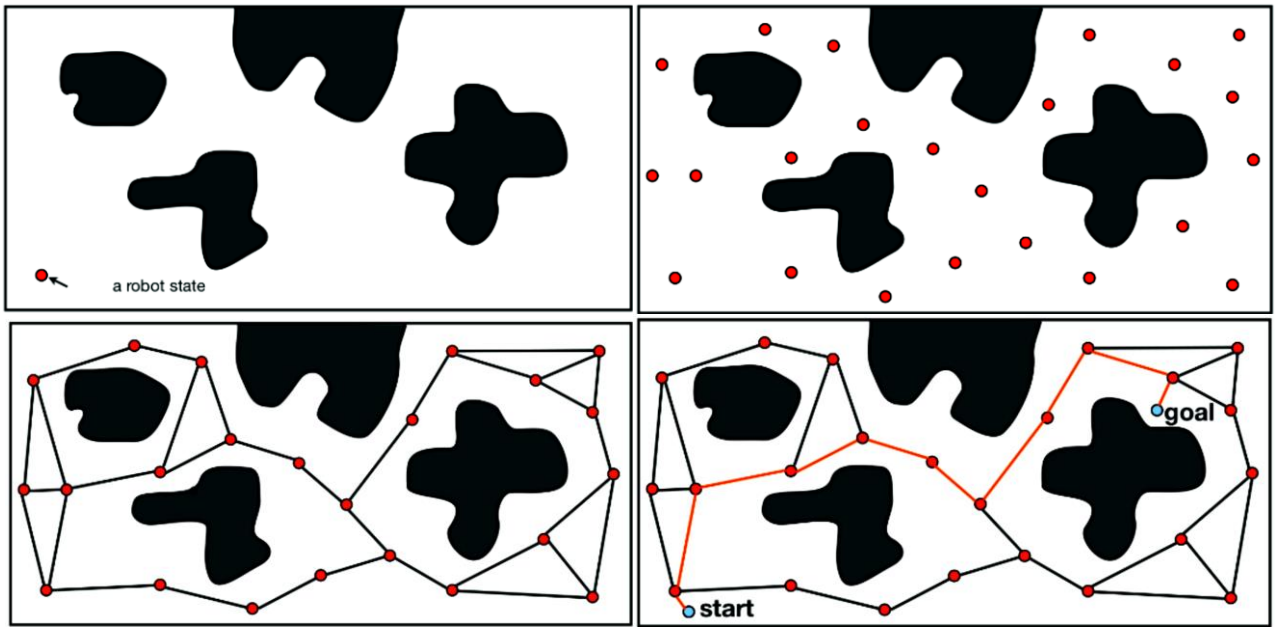


Figura 28 – Generazione di una RoadMap con metodo PRM. Di conseguenza vengono mostrati i passaggi per la generazione di un percorso valido per il passaggio da una configurazione q_{start} ad una q_{goal}. Fonte ([12] - Kavraki Lab Rice University,, 2021)

Algoritmo *A-star*

L'algoritmo *A** o anche *A-star* è un algoritmo di ricerca su grafi che individua un percorso da un nodo iniziale verso un nodo destinazione. Questo algoritmo utilizza una "stima euristica" che classifica ogni nodo della mappa attraverso una strada migliore che passi attraverso tale nodo.

Sono presenti:

- Due liste:
  - Una lista ordinata denominata *"OPEN"* che contiene i nodi per cui l'esplorazione deve ancora avvenire. Questa lista di elementi è un elenco ordinato funzione del costo totale stimato per raggiungere il nodo destinazione;
  - Una lista denominata *"CLOSED"* che contiene i nodi per cui l'esplorazione ha già avuto luogo.
- Una matrice denominata *"cost[node1,node2]"* che codifica l'insieme dei lati. Un valore positivo corrisponde al costo minimo per muoversi dal *node1* → *node2*. Un valore negativo indica che non esistono percorsi ammissibili per muoversi da un nodo all'altro.
- Un vettore "*past_cost[node]"* contiene il costo minimo per raggiungere il nodo partendo dal nodo di partenza.
- L'albero di ricerca è definito da un vettore *"parent[node]"* che contiene, per ogni nodo, un collegamento con il precedente connesso al percorso più breve trovato fino a quel momento. Partendo dallo *start_node* → *nodo CURRENT*.

Per inizializzare la ricerca la matrice *cost[]* è costruita per codificare i lati. La lista "*OPEN*" è inizializzata con lo *start_node=1*, il costo per raggiungere tale nodo è pari a $past\_cost[1] = 0$, mentre con $node \in \{2, \dots, N\}, past\_cost[node] = \infty$, indicando il fatto che in quel momento non si ha idea del costo necessario per raggiungere tali nodi.

Ad ogni iterazione dell'algoritmo, il primo nodo della lista *"OPEN"* viene rimosso da *"OPEN"* e viene chiamato *CURRENT*. Il nodo *CURRENT* viene aggiunto alla lista *"CLOSED"*. N.B.: il primo nodo della lista *"OPEN"* è quello che minimizza il costo del percorso per raggiunger il nodo destinazione passando per quel nodo. A questo punto viene calcolato il $est\_total\_cost[node]$ come:

$$est\_total\_cost[node] = past\_cost[node] + heuristic\_cost\_to\_go[node] \quad (3.5)$$

Con $heuristic\_cost\_to\_go[node] \geq 0$ si stima un valore ottimistico (sottostimato) del costo per arrivare al nodo destinazione partendo dal nodo *"node"*.

Se il nodo *CURRENT* è nell'obiettivo prefissato, allora la ricerca è terminata e il percorso è ricostruito.

Se il nodo *CURRENT* non è il nodo destinazione, allora, per ogni nodo vicino *(neighbour=nbr)* di *CURRENT* nel grafo, che però non appartiene all'elenco *"CLOSED"*, viene calcolato il:

$$tentative\_past\_cost[nbr] = past\_cost[current] + cost[current, nbr] \quad (3.6)$$

Se $tentative\_past\_cost[nbr] < past\_cost[current]$ allora il nodo *nbr* può essere raggiunto con un costo inferiore rispetto a quello che si pensava precedentemente. $past\_cost[nbr]$ viene impostato come *CURRENT*. Il nodo *nbr* viene aggiunto all'elenco *"OPEN"* tenendo conto della stima del suo costo totale.

L'algoritmo ritorna all'inizio del ciclo principale, rimuovendo il primo nodo dall'elenco *"OPEN"*, chiamandolo *CURRENT*.

Se *"OPEN"* è un insieme vuoto e non si è giunti ad una soluzione nella ricerca, allora, non esiste una soluzione allo specifico problema e l'algoritmo restituirà un fallimento.

L'algoritmo *A** è garantito per restituire il percorso minimo dato che i nodi vengono controllati per l'inclusione nella destinazione impostata solo quando hanno il costo totale minimo stimato tra tutti i nodi.

Fonte ([11] - Kevin M. Lynch and Frank C.Park,, 2017) "Modern Robotics: Mechanics, Planning and Control.".

Metodo *PRMstar*

Utilizzando il classico metodo *PRM* per pianificare un percorso in un ambiente complesso, ci sono due problematiche da risolvere:

- la presenza di pochi nodi per costruire un percorso valido nelle regioni critiche. Infatti, non è garantita una buona connettività di C_{free}, nelle regioni di spazio strette e/o limitate a causa della presenza di ostacoli;
- la presenza di pochi nodi nelle regioni di confine.

Per ovviare a queste problematiche si adotta la seguente strategia:

A. Strategia di campionamento ibrida

La strategia di campionamento ibrida prevede:

- un test di campionamento a ponte che permette di incrementare i nodi nei passaggi stretti;

- un campionamento non uniforme che permette di aumentare i nodi nelle regioni al bordo.

Di seguito è riportata un'immagine che graficamente fornisce un'idea dei campionamenti impiegati per migliorare la densità dei nodi in prossimità di passaggi stretti e delle configurazioni al bordo.

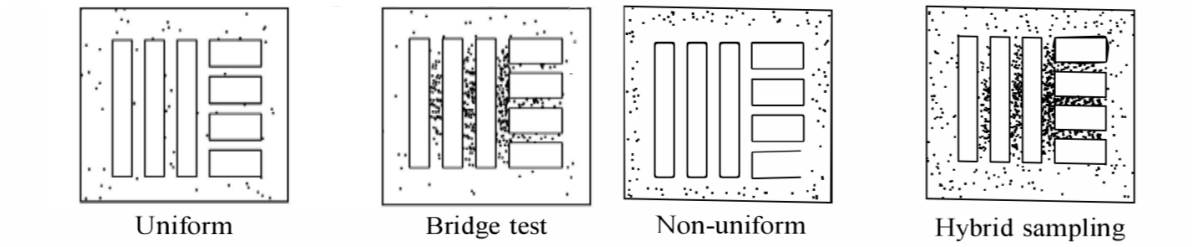


Figura 29 - Densità dei nodi in prossimità degli spazi stretti e in prossimità delle configurazioni di bordo risulta aumentata a valle della strategia di campionamento ibrida. Fonte ([13] - Fang Yuan, Liang, Jia-Hong; Fu, Yue-Wen; Xu, Han-Cheng; Ma, Ke, 2015).

B. Pianificazione della *RoadMap*

Il compito del pianificatore della *RoadMap* è di connettere i nodi e costruire un certo numero di segmenti e movimenti privi di collisione.

C. Pianificazione del percorso

L'algoritmo *A-star* restituirà il percorso più breve della *RoadMap* generata.

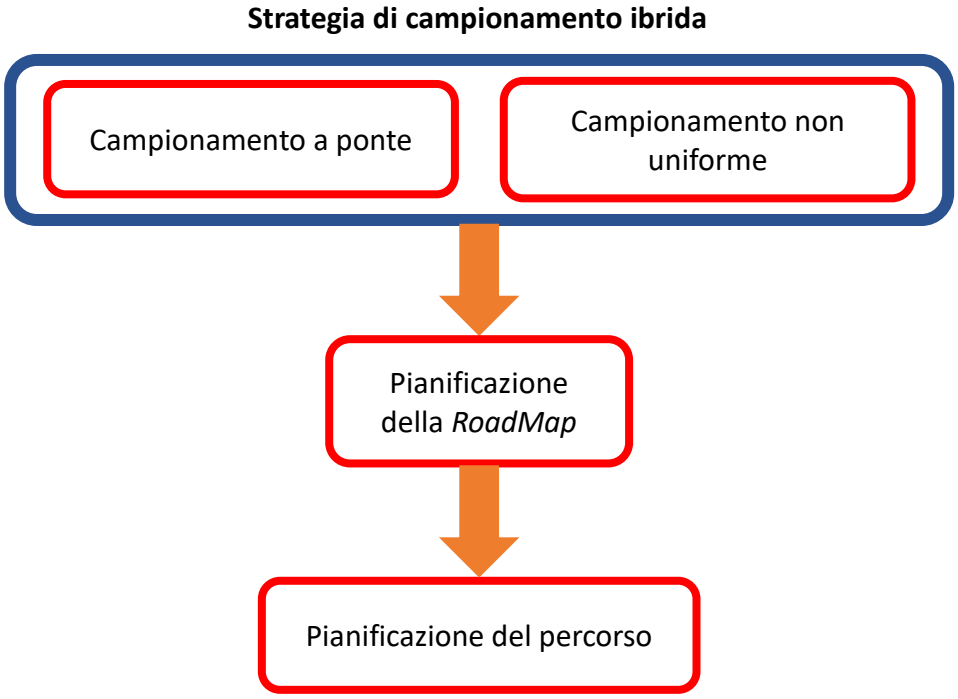


Figura 30 - Passaggi della pianificazione PRMstar.

Fonte ([13] - Fang Yuan, Liang, Jia-Hong; Fu, Yue-Wen; Xu, Han-Cheng; Ma, Ke, 2015).

Metodo *RRT (Rapidly-exploring Random Tree)*

L'algoritmo *RRT* (acronimo di *Rapidly-exploring Tandom Tree*) cerca una soluzione di moto privo di collisioni per muovere il robot da una configurazione $\boldsymbol{q_{start}}$ ad una configurazione $\boldsymbol{q_{goal}}$.

Questo tipo di pianificatore del moto viene impiegato per risolvere rapidamente problemi complessi di pianificazione del moto malgrado la scarsa qualità della ricerca.

L'algoritmo *RRT* sfrutta il campionamento dello spazio delle configurazioni (C_{free}) per generare un grafo in cui non sono presenti cicli. Un grafo connesso in cui non sono presenti cicli è detto albero. In un albero, ciascun nodo ha al massimo un nodo genitore. Un albero ha sempre un nodo radice (privo di collegamenti con un nodo genitore) ed un certo numero di nodi foglia che non presentano collegamenti con nodi figli.

L'algoritmo *RRT* inizia la ricerca dell'albero ***T*** partendo dal nodo $\boldsymbol{q_{start}}$ (nodo radice nonché lo stato attuale del robot) all'interno dello spazio delle configurazioni.

Ad ogni iterazione/campionamento viene individuato un nodo $\boldsymbol{q_{rand}}$ (nodo campionato dallo spazio delle configurazioni ammissibili C_{free}). A valle di questo campionamento si individua $\boldsymbol{q_{nearest}}$**:** il nodo più vicino a $\boldsymbol{q_{rand}}$ che però già appartiene all'albero ***T***.

A questo punto viene richiamato un pianificatore locale in modo da verificare effettivamente la possibilità di generare un percorso valido tra il nodo $\boldsymbol{q_{nearest}}$ e lo stato campionato $\boldsymbol{q_{rand}}$**.**

Se il percorso risulta privo di collisioni allora verrà aggiunto all'albero ***T***, $\boldsymbol{q_{new}}$**,** con un ramo (linea retta) di collegamento pari ad una lunghezza "delta" δ da $\boldsymbol{q_{nearest}}$ a $\boldsymbol{q_{new}}$. Mediante un test di collisione, si verifica che tanto $\boldsymbol{q_{new}}$ quanto il segmento da $\boldsymbol{q_{near}}$ a $\boldsymbol{q_{new}}$ appartengono a C_{free}.

Si fa presente che $\boldsymbol{q_{rand}}$ non verrà aggiunto all'albero, e quindi non è sottoposto ad una verifica di collisione. Infatti, la sua unica funzione è quella di indicare una direzione di espansione di ***T***.

Ad ogni campionamento/iterazione si avrà un nuovo punto, che a sua volta genererà un nuovo ramo, collegando il nuovo punto al nodo dell'albero più vicino. L'albero continuerà ad ampliarsi nello spazio delle configurazioni finché il *random point* non è proprio il goal *node*.

Il processo iterativo si arresterà nel momento in cui $\boldsymbol{q_{new}}$ si troverà all'interno della regione obiettivo Q_{goal}. Una volta raggiunto il nodo goal, si genererà il percorso che collega il nodo di start con il nodo goal. Fonte ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

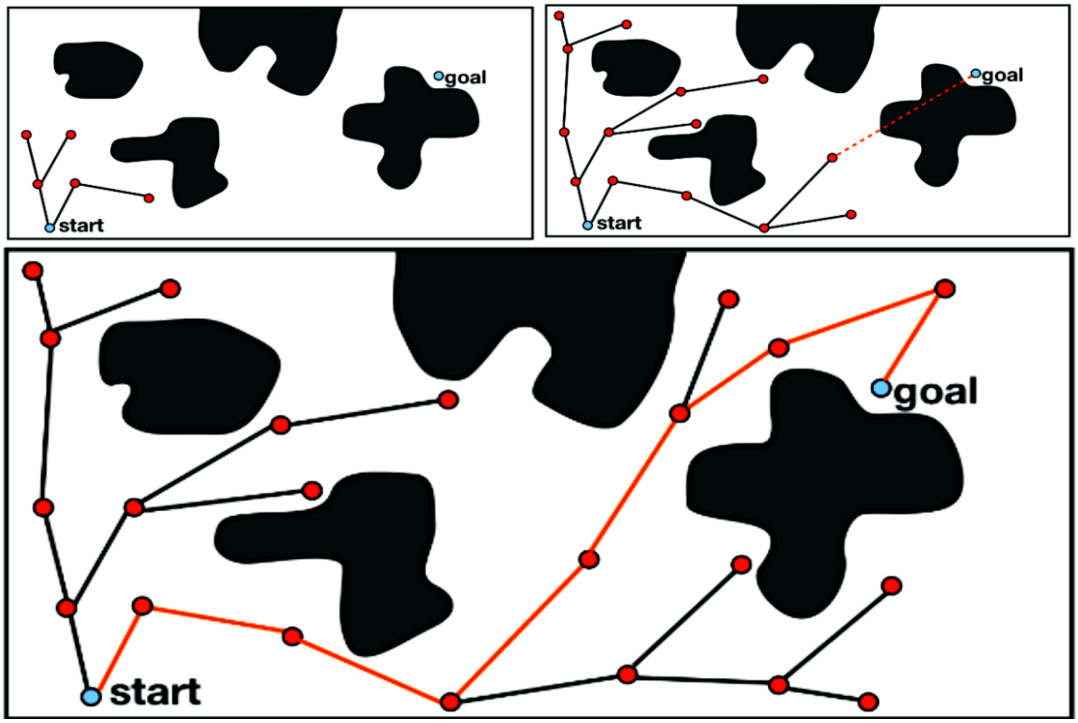


Figura 31 - Generazione di un albero tramite il metodo RRT. Di conseguenza vengono mostrati i passaggi per la realizzazione di un percorso valido per il passaggio da una configurazione q_{start} ad una q_{goal}. Fonte ([12] - Kavraki Lab Rice University,, 2021)

Metodo *RRT Connect* (RRTBidirezionale)

Il metodo *RRTConnect,* o anche RRTBidirezionale, è un metodo che ottimizza la ricerca di un cammino da $\boldsymbol{q_{start}}$ a $\boldsymbol{q_{goal}}$ rispetto al metodo *RRT*. Nel metodo bidirezionale vengono implementati due alberi: detti ***Ts*** (albero di partenza) e ***Tg*** (albero destinazione), aventi radice rispettivamente in $\boldsymbol{q_{start}}$ e $\boldsymbol{q_{goal}}$.

Ad ogni iterazione entrambi gli alberi vengono espansi seguendo l'algoritmo di espansione descritto dal metodo *RRT*.

L'algoritmo prevede che dopo un certo numero di iterazioni, si tenta di connettere i due alberi **Ts** e **Tg** "estendendoli" l'uno verso l'altro.

Il tentativo di connessione dei due alberi avviene generando una $\boldsymbol{q_{new}}$ attraverso un'espansione su ***Ts***, e, cercando di estendere l'altro albero ***Tg*** fino a $\boldsymbol{q_{new}}$.

Per realizzare tale estensione la procedura di espansione definita precedentemente viene modificata e in particolare, si utilizza la configurazione $\boldsymbol{q_{new}}$ come $\boldsymbol{q_{rand}}$. A questo punto si individua la configurazione $\boldsymbol{q_{near}}$ a essa più vicina in ***Tg*** e ci si muove in linea retta da $\boldsymbol{q_{near}}$ cercando di raggiungere $\boldsymbol{q_{rand}}(=\boldsymbol{q_{new}})$, e quindi non solo di un passo "delta" δ.

Il segmento da $\boldsymbol{q_{near}}$ a $\boldsymbol{q_{new}}$ è sottoposto ad un test di collisione. Se non si verificano collisioni, l'estensione è completa e i due alberi sono stati connessi; altrimenti viene ritagliata una porzione del segmento priva di collisioni, e la si aggiunge a ***Tg*** insieme al suo punto esterno.

A questo punto, si scambiano di ruolo ***Tg*** e ***Ts***, e si prosegue con il tentativo di connessione.

Se dopo un certo numero di iterazioni il tentativo di connessione dei due alberi non ha successo, allora significa che i due alberi sono ancora molto lontani e quindi l'algoritmo riparte dalla fase di espansione.

Fonte ([10] - Robotica. modellistica, pianificazione e controllo, 2008).

Metodo *RRTstar*

L'algoritmo di pianificazione *RRT* base restituisce un successo una volta che il moto verso Q_{goal} è stato trovato. Dato che i lati dell'albero non vengono mai cancellati o cambiati, l'algoritmo RRT solitamente non converge ad una soluzione ottimale.

Il metodo *RRT** o *RRT-star* è un modo alternativo per il raggiungimento di una soluzione del problema di pianificazione. Questo metodo continua a far girare l'algoritmo di ricerca anche dopo che una soluzione è stata trovata. La ricerca termina solo quando è stata raggiunta la condizione di tempo massimo concesso al codice per la ricerca del percorso. A questo punto il percorso con costo minimo viene restituito dal pianificatore come soluzione al problema di pianificazione. In questa maniera la soluzione *RRT* può continuare a migliorarsi nel tempo.

L'algoritmo *RRT-star* è una variante dell'algoritmo a singolo albero *RRT* che continuamente cerca nuove connessioni all'interno dell'albero in modo da garantire che questo restituisca sempre il percorso più breve dal nodo $\boldsymbol{q_{start}}$ ad ogni nodo dell'albero.

Per modificare l'algoritmo da *RRT* a *RRT-star*, ad ogni iterazione vi è un test, da parte del pianificatore locale, tale per cui vengono analizzati tutti i nodi $\boldsymbol{q} \in Q_{near}$ che:

1 – presentino un movimento privo di collisioni;

2 – minimizzino il costo totale di movimento nel percorso da $\boldsymbol{q_{start}}$ a $\boldsymbol{q_{new}}$ e quindi non viene verificato esclusivamente il costo del nuovo segmento aggiunto. Il costo totale è il costo per raggiungere il nodo candidato $\boldsymbol{q} \in Q_{near}$ più il costo del nuovo segmento.

Il passaggio successivo è considerare ciascun $\boldsymbol{q}$ appartenente a Q_{near} per verificare se potrebbe essere raggiunto con un costo di movimento inferiore attraverso $\boldsymbol{q_{new}}$.

Se dovesse essere così, il nodo $\boldsymbol{q}$ verrebbe sostituito dal nodo $\boldsymbol{q_{new}}$.

In questo modo, l'albero è riconnesso in modo incrementale per eliminare alti costi di movimento a favore del minimo costo di moto disponibile.

La definizione di Q_{near} dipende dal numero di campionamenti nell'albero, dettagli del metodo di campionamento, dimensioni dello spazio di ricerca, ed altri fattori.

A differenza di *RRT*, la soluzione restituita da *RRT** si avvicina alla soluzione ottimale all'aumentare dei nodi campionati. Come per *RRT*, l'algoritmo *RRT** è completo in probabilità.

Fonte ([11] - Kevin M. Lynch and Frank C.Park,, 2017) "Modern Robotics: Mechanics, Planning and Control.".

3.2. Pianificazione del moto in *MoveIt*

Richiesta di Pianificazione del moto

La richiesta di pianificazione del moto definisce quello che il pianificatore del moto deve fare.

Solitamente quello che viene richiesto al pianificatore del moto è di far muovere un braccio robotico in una certa configurazione, oppure di muovere l'organo terminale verso una nuova posa.

La pianificazione del moto all'interno di *MoveIt* è gestita dal nodo `move_group`. Il nodo `move_group` è il nodo fondamentale in *MoveIt* perché è colui il quale si occupa di gestire le informazioni da pubblicare e sottoscrivere dai *Topic*, dalle *Action* e dai *Service* per la pianificazione e la successiva esecuzione di una traiettoria da parte del manipolatore.

L'oggetto `planning_pipeline` funge da interfaccia tra il nodo `move_group` e i pianificatori dei percorsi. Attraverso un'*Action* o un *Service* di *ROS* (messo a disposizione del nodo `move_group`) viene inviata all'oggetto `planning_pipeline` la richiesta per la ricerca di un percorso valido per portare il manipolatore da una configurazione $\boldsymbol{q_{start}}$ a una $\boldsymbol{q_{goal}}$. I pianificatori di percorsi predefiniti in *MoveIt* sono quelli offerti da *OMPL*. *OMPL (*acronimo di *Open Motion Planning Library)* è una libreria di pianificazione del moto *open-source* che principalmente implementa pianificatori di movimento probabilistici basati su campionamento (il principio di funzionamento di questi pianificatori è stato affrontato nel paragrafo 3). Mentre, attraverso la libreria *FCL (Flexible Collision Library)* vi è la rappresentazione degli oggetti di collisone all'interno dello spazio delle configurazioni.

Da riga di comando, è possibile specificare il pianificatore da impiegare per la richiesta di pianificazione del moto.

In fase di pianificazione del moto, *MoveIt*, garantisce il rispetto di alcuni vincoli cinematici come ad esempio:

- Vincoli di posizione: limitano la posizione di un link in modo che si trovi all'interno di una regione di spazio;
- Vincoli di orientamento: limitano l'orientamento di un link in modo che questo rientri nei limiti di rollio, beccheggio o imbardata specificati;
- Vincoli di visibilità: limitano un punto appartenente ad un link affinché esso si trovi all'interno del cono di visibilità di un particolare sensore;
- Vincoli fisici di giunto: limitano un giunto a trovarsi tra due valori limite;

Al termine della pianificazione del moto da parte del nodo `move_group`, verrà generata la traiettoria desiderata in funzione della rispettiva richiesta. Questa traiettoria farà in modo da muovere il manipolatore (o alcuni giunti) fino alla posizione desiderata. Si fa presente che il risultato

che esce dal `move_group` sarà la traiettoria e non il percorso. `move_group` utilizzerà le velocità e le accelerazioni specificate all'interno del file `joint_limits.yaml` per generare una traiettoria che rispetti i vincoli di velocità e accelerazione dei giunti.

Motion Planning Pipeline: Pianificatori del moto e *Planning Request Adapters*

La procedura completa di pianificazione del moto concatena un pianificatore del percorso con altri componenti chiamati *Planning Request Adapters*. I *Planning Request Adapters* consentono la pre-elaborazione delle richieste di pianificazione del moto e la post-elaborazione dei *feedback* provenienti dalla pianificazione.

MoveIt fornisce una serie di *Motion Planning Adapters* (che verranno di seguito esplicitati), ciascuno dei quali esegue una funzione specifica.

FixStartStateBounds - Correttore dei limiti dello stato di avvio

L'adattatore *"FixStartStateBounds"* (Correttore dei limiti dello stato di avvio) si occupa di correggere lo stato di avvio del manipolatore, in modo che questo, rientri nei limiti di giunto specificati nell'*URDF*. La necessità di questo adattatore si presenta nel momento in cui, i limiti di giunto per il robot fisico, non sono configurati correttamente. Il robot potrebbe trovarsi in una configurazione in cui uno, o più giunti, si trovano al di fuori dei limiti di giunto. In questo caso, il pianificatore del moto, il nodo `move_group`, non sarà in grado pianificare, poiché non riconoscerà lo stato di partenza, in quanto al di fuori dei limiti di giunto.

L'adattatore *"FixStartStateBounds"* si occuperà eventualmente di aggiustare lo stato iniziale dei giunti spostandoli verso i limiti di giunto più vicini.

FixStartStateCollision - Correttore di collisione dello stato di avvio

L'adattatore *"FixStartStateCollision"* (correttore di collisione dello stato di avvio) viene impiegato nel momento in cui viene riscontrata una configurazione iniziale in collisione. Questo adattatore tenta di campionare una nuova configurazione, priva di collisioni, molto vicina alla configurazione che presenta una collisione, variandone di poco i valori di giunto.

FixStartStatePathConstraints - Correttore dei vincoli di percorso dello stato di avvio

L'adattatore *"FixStartStatePathConstraints"* (correttore dei vincoli di percorso dello stato di avvio) viene richiamato quando lo stato iniziale del manipolatore non rispetta i vincoli di percorso specificati. Questo adattatore tenterà di pianificare un percorso tra la configurazione attuale del robot fino ad una nuova posizione in cui viene rispettato il vincolo di percorso.

La nuova configurazione/posizione servirà come stato iniziale per la pianificazione.

AddTimeParameterization - Parametrizzazione del percorso nel tempo

In uscita dal pianificatore di percorsi si ha un percorso costituito da una serie di punti di via collegati tra loro da segmenti lineari. I punti di via, però, sono punti di non derivabilità nel percorso.

Risulta quindi necessario raccordare i segmenti del percorso al fine di ottenere un'esecuzione fluida della traiettoria. Il raccordo di tali segmenti è affidato a degli algoritmi che verranno presentati di seguito. Questi algoritmi si occupano, inoltre, di generare una legge temporale da imporre ai giunti del manipolatore.

L'adattatore *"AddTimeParameterization"* (aggiunta della parametrizzazione nel tempo) ha il compito di "parametrizzare" il percorso nel tempo applicando vincoli di velocità e accelerazione.

Si rammenda che, in fase di parametrizzazione nel tempo, il nodo `move_group`, impone i limiti di velocità e accelerazione presenti all'interno del file `joint_limits.yaml`. Questi valori verranno sovrascritti ai limiti di velocità e accelerazione presenti all'interno del file *URDF*.

Di seguito verranno presentate due possibili tecniche impiegate per la generazione di una traiettoria in *MoveIt*:

1. *Iterative Parabolic Time Parameterization (IPTP)* - Algoritmo iterativo di parametrizzazione del tempo parabolico;
2. *Time-Optimal Trajectory Generation (TOTG)* - Generazione di traiettorie ottimali nel tempo.

Per lo studio di questo elaborato è stato impostato un algoritmo iterativo di parametrizzazione del tempo parabolico *(IPTP).*

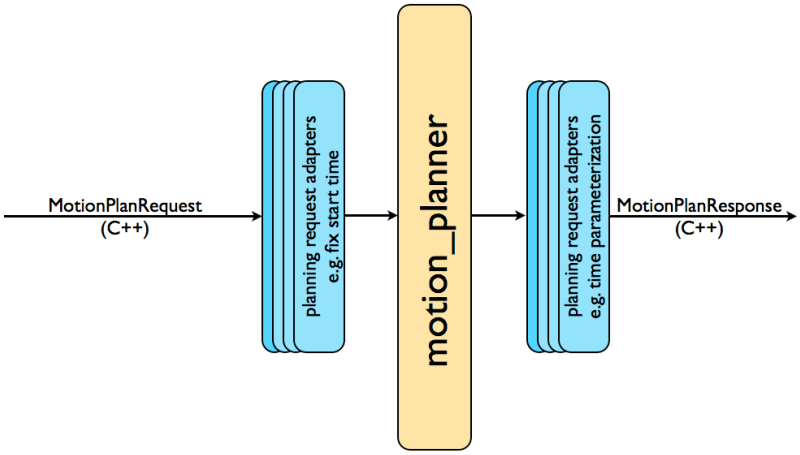


Figura 32 – Passaggi che partono dalla richiesta di pianificazione del moto fino all'esito della pianificazione stessa da parte del pianificatore del moto. Fonte ([8] - MoveIt!, s.d.)

Sequenza di polinomi parabolico-lineari in prossimità dei punti di via

In molti metodi esposti in letteratura, si presuppone che per i percorsi, siano noti gli intervalli di tempo necessari a compiere ciascun tratto lineare.

L'algoritmo utilizzato da *MoveIt* è in grado di generare una traiettoria che non prevede il passaggio per tutti i punti di via del percorso, ma, vi è un raccordo parabolico tra due segmenti successivi del percorso che presentano differente coefficiente angolare. Così facendo, la traiettoria seguirà il percorso ammettendo una certa tolleranza di errore in prossimità dei punti di via, e il manipolatore si muoverà in maniera fluida, rispettando i vincoli di velocità e accelerazione dei suoi giunti.

Ipotizzando che la traiettoria sia definita da una durata pari a t_f e da una funzione $q : [0, t_f] \rightarrow C_{free}$, che fornisce le configurazioni del manipolatore in ogni istante di tempo.

Le condizioni al contorno sono:

- $q(0) = w_1, q(t_f) = w_n, \dot{q}(0) =, \dot{q}(t_f) = 0;$
- $\forall t : |\dot{q}(t)| \leq v_{max} \wedge |\ddot{q}(t)| \leq a_{max}$

Attraverso una procedura iterativa, l'algoritmo associa un istante di tempo ai punti di via del percorso $i = 0, \dots, n$ e definisce il tempo necessario per l'esecuzione della traiettoria.

Per la parametrizzazione del percorso nel tempo, si presuppone che il movimento del manipolatore con *k-DOF* venga eseguito sfruttando la massima velocità $v_{max}[j]$ e accelerazione $a_{max}[j]$ per ciascun giunto $j = 1, \dots, k$.

In prima battuta vengono scelti gli istanti di tempo dei vari punti di via in modo che la velocità per l'esecuzione della traiettoria sia massimizzata, seguendo la relazione:

$$dt_{i,j_{min}} = \max_j \frac{|q_{i+1}[j] - q_i[j]|}{v_{max}[j]} \qquad (3.7)$$

A questo punto la velocità imposta a ciascun giunto viene calcolata come segue:

$$v_{i,j} = \frac{|q_{i+1}[j] - q_i[j]|}{dt_i} \qquad (3.8)$$

Dato che il valore di $dt_{i,j_{min}}$ viene calcolato come il massimo valore tra gli intervalli temporali necessari ai giunti per passare dalla configurazione $q_i[j]$ alla configurazione $q_{i+1}[j]$, la successiva definizione delle velocità dei singoli giunti $v_{i,j}$, garantisce la condizione $v_{i,j} \leq v_{max}[j]$, $\forall j = 1, \dots, k$.

A valle della definizione dei valori di velocità da attribuire a ciascun giunto per l'esecuzione del percorso, vengono calcolati i rispettivi valori di accelerazione come segue:

$$a_{i,j} = 2 \cdot \frac{|v_{i+1}[j] - v_i[j]|}{dt_{i+1} + dt_i} \qquad (3.9)$$

Se uno dei giunti supera il suo limite di accelerazione lungo l'i-esimo tratto del percorso, attraverso una procedura iterativa, verrà incrementato il corrispettivo intervallo temporale dt_i di una piccola frazione, finché non verranno rispettati i limiti di velocità e accelerazione di tutti i giunti.

Nell'immagine che segue è riportato l'esempio di una traiettoria e dei rispettivi comandi di velocità che verranno inviati al controller del manipolatore per l'esecuzione della stessa.

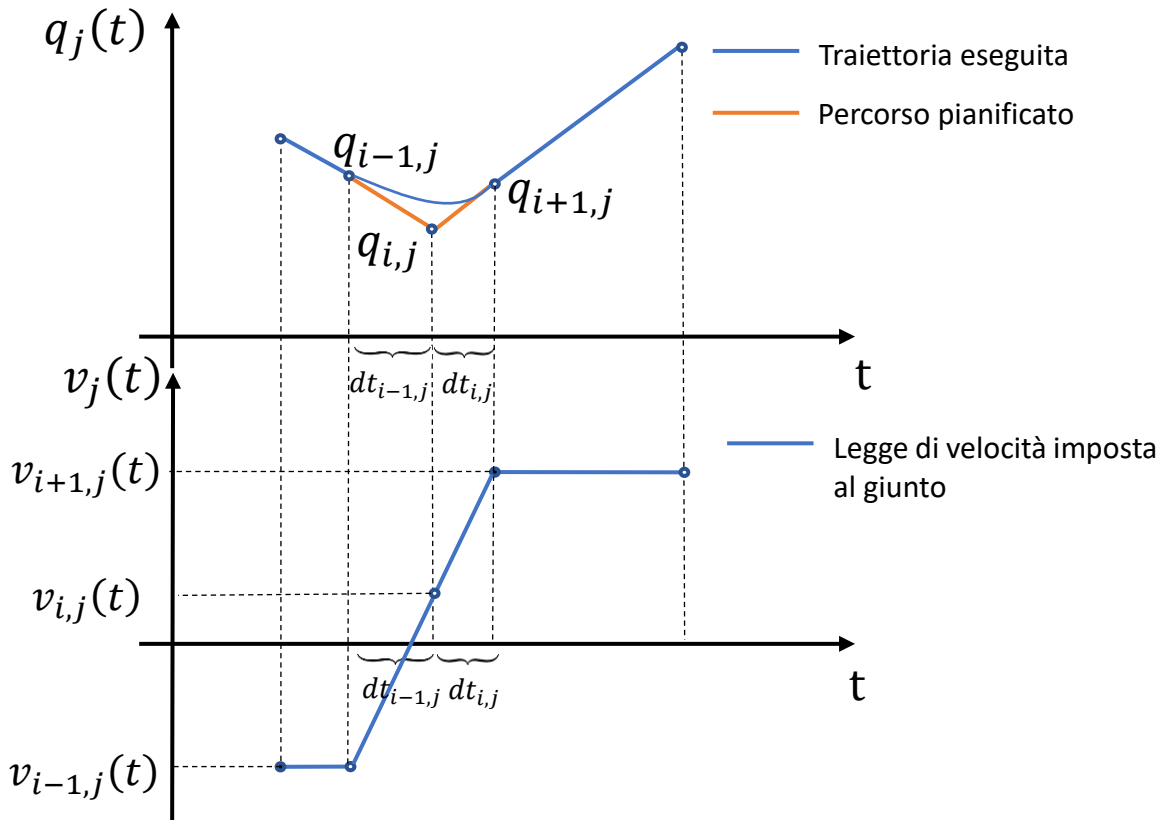


Figura 33 - Nel grafico in alto è rappresentato un esempio di traiettoria generata per un generico giunto j. Nel grafico in basso è riportato il set di velocità imposto allo stesso giunto per l'esecuzione della rispettiva traiettoria.

Generazione di traiettorie ottimali nel tempo

L'algoritmo *Time Optimal Trajectory Generation (TOTG)* è un algoritmo di parametrizzazione nel tempo impostabile su *MoveIt*.

Anche questo algoritmo presenta una soluzione al problema della generazione di traiettoria lungo un percorso definito rispettando i limiti di velocità e accelerazione dei vari giunti.

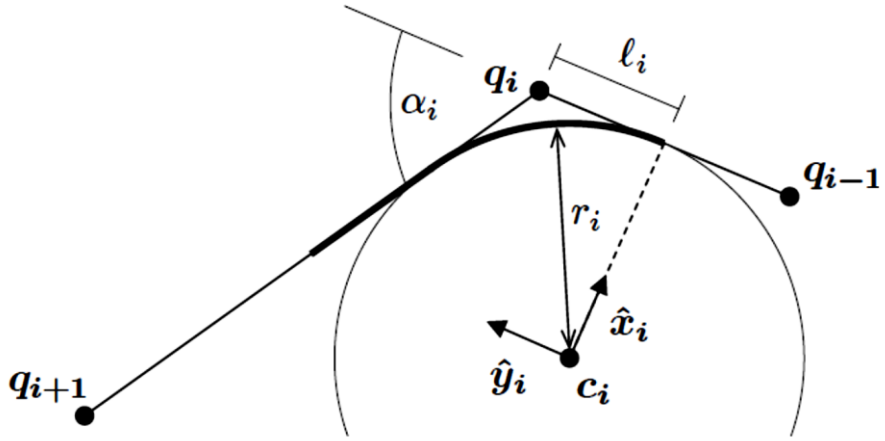


Figura 34 - Raccordo circolare tra due segmenti successivi fi un percorso. Fonte ([21] - Tobias Kunz and Mike Stilman - Georgia Institute of Technology)

Il metodo che verrà di seguito descritto prevede che in prossimità dei punti di via (punti di non derivabilità) vi sia un raccordo circolare tra il segmento precedente e quello successivo.

L'obiettivo è quello di trovare un segmento circolare che parta tangenzialmente a ciascun segmento lineare e finisca tangenzialmente al segmento lineare successivo. Inoltre, per come è strutturato l'algoritmo, è garantito che il segmento circolare non sia più della metà di ciascun segmento lineare adiacente. Viene inoltre garantito uno scostamento massimo della traiettoria rispetto al percorso nominale.

Dato che la soluzione al problema è vincolata a seguire il percorso generato dai pianificatori *RRT* o *PRM*, il problema viene ridotto ad un problema ad una sola dimensione.

Il percorso definito come $f : [0, s_f] \rightarrow \mathbb{R}^n$ e la velocità in ogni punto del percorso definita come $\dot{s} = \frac{ds}{dt}$.

La generica configurazione $\boldsymbol{q}$ in un punto s del percorso è data dalla relazione:

$$0 \leq s \leq s_f \qquad \boldsymbol{q} = \boldsymbol{f}(s) \qquad (3.10)$$

Le rispettive velocità e accelerazioni saranno:

$$\dot{\boldsymbol{q}} = \frac{\boldsymbol{d}}{\boldsymbol{dt}}\boldsymbol{f}(s) = \frac{\boldsymbol{d}}{ds}\boldsymbol{f}(s)\frac{ds}{dt} = \boldsymbol{f}'(s)\dot{s} \qquad (3.11)$$

$$\ddot{\boldsymbol{q}} = \boldsymbol{f}'(s)\ddot{s} + \boldsymbol{f}''(s)\dot{s}^2 \qquad (3.12)$$

Lungo il tratto circolare, s è l'arco, $\dot{s}$ e $\ddot{s}$ sono la velocità e l'accelerazione lungo il percorso pertanto, $\boldsymbol{f}'(s)$ sarà il vettore tangente al percorso e $\boldsymbol{f}''(s)$ sarà la curvatura.

Per poter ridurre il problema di generazione della traiettoria ad un problema in una sola dimensione e, sfruttando le relazioni sopra riportate, i limiti di accelerazione e velocità dei giunti vengono trasformati in vincoli di velocità lungo il percorso $\dot{s}(s)$ e in vincoli di accelerazione lungo il percorso $\ddot{s}(\dot{s}, s)$.

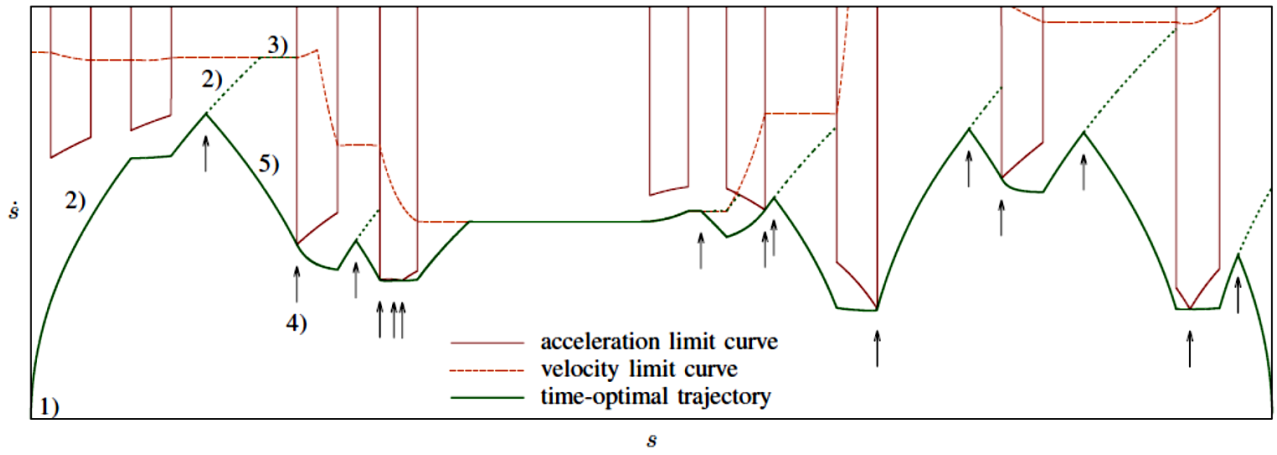


Figura 35 - Diagramma delle fasi s-$\dot{s}$ si una generica traiettoria. Fonte ([21] - Tobias Kunz and Mike Stilman - Georgia Institute of Technology)

Il grafico sopra riportato è noto come piano delle fasi ed è un grafico s-$\dot{s}$ che mette in relazione la velocità in funzione del percorso. Il punto di partenza $s(0)$ è in basso a sinistra, il punto di arrivo $s(s_f)$ è in basso sulla destra.

La curva tratteggiata di colore arancione è la curva limite della velocità $\dot{s}_{max}$ che ovviamente sarà funzione dei limiti di velocità $\dot{s}_{vel}^{max}(s)$ e accelerazione $\dot{s}_{acc}^{max}(s)$. Ovviamente la traiettoria generata, nonché la curva verde, dovrà trovarsi sempre al di sotto di tale curva limite. L'obiettivo desiderato è che la curva verde sia quanto più vicina possibile a quella di colore arancione in ogni punto del percorso, ma, allo stesso tempo l'algoritmo prevede che l'accelerazione del percorso sia sempre ad uno dei suoi limiti ovvero: $\ddot{s}_{min}$ oppure $\ddot{s}_{max}$. Queste condizioni devono essere rispettate per tutto il diagramma delle fasi.

L'algoritmo individua i punti di commutazione in prossimità del passaggio da accelerazioni minima ad accelerazione massima e viceversa. Nell'immagine sopra riportata, i punti di commutazione sono individuati dalle frecce.

Fonte ([21] - Tobias Kunz and Mike Stilman - Georgia Institute of Technology).

Controller Interface

Per l'esecuzione dei movimenti e delle traiettorie, *ROS*, mette a disposizione il pacchetto `ros_control`. `move_group` comunica con il controller del robot attraverso l'interfaccia di tipo *Action: FollowJointTrajectory*. Un server sul robot fornisce questa *Action*. `move_group` crea un'istanza di un *client* per comunicare con il *ControllerActionServer* del robot.

Attraverso l'*Action FollowJointTrajectory* vengono specificati gli obiettivi e le tolleranze per l'esecuzione delle traiettorie.

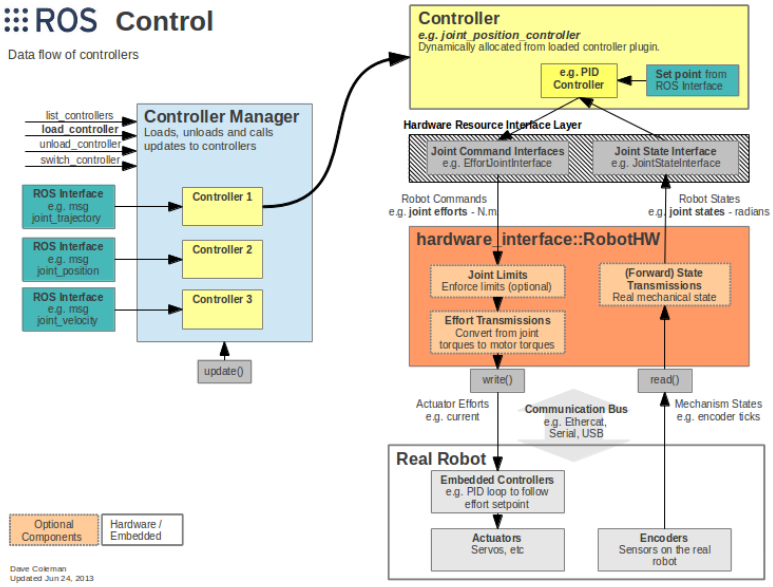


Figura 36 – Diagramma di flusso dei dati di set e feedback gestiti dal nodo ros_control. Fonte ([26] - ROS Control, s.d.)

Planning Scene

Planning Scene è utilizzato per rappresentare l'ambiente di lavoro intorno al robot.

Planning Scene Monitor è gestito dal nodo `move_group`, e si sottoscrive a diversi topic per ricevere informazioni sullo stato del robot (attraverso il *Topic* `/j2s7s300_driver/out/joint_state`) e da eventuali sensori e sistemi di visione in grado di fornire informazioni utili al manipolatore.

L'utente può anche decidere di simulare l'ambiente di lavoro all'interno del quale il robot dovrà lavorare pubblicando la scena di pianificazione sul *Topic* `/planning_scene`.

World Geometry Monitor

World Geometry Monitor costruisce la geometria del mondo virtuale in cui è immerso il robot utilizzando informazioni dai sensori e dagli input forniti dall'utente.

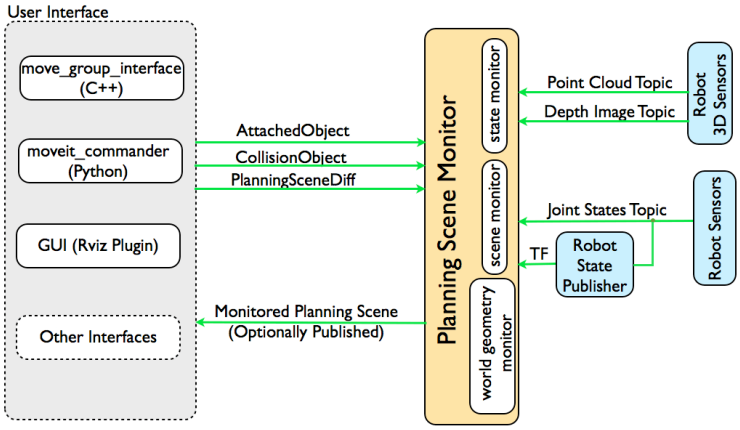


Figura 37 – Informazioni che vengono prelevate e scambiate dal nodo planning_scene nella gestione del robot. Fonte ([8] - MoveIt!, s.d.)

Controllo di Collisione

Il controllo di collisione in *MoveIt* è configurato all'interno del *Planning Scene* utilizzando l'oggetto *CollisionWorld. MoveIt* è configurato in modo che gli utenti non debbano mai occuparsi di

come avviene il controllo di collisione. Il controllo delle collisioni in *MoveIt* viene eseguito richiamando la libreria *FCL (*acronimo di *Flexible Collision Library)*.

La libreria *FCL* integra diverse tecniche al fine di garantire controlli di collisione e calcoli di prossimità in modo rapido e preciso.

È basata sulla rappresentazione gerarchica degli oggetti e garantisce: il rilevamento discreto di collisione, il rilevamento continuo di collisione, il calcolo della distanza di separazione e fornisce anche una stima della profondità di penetrazione. Le informazioni relative alla configurazione degli oggetti e alla loro rappresentazione gerarchica, sono memorizzate all'interno di una struttura chiamata `CollisionObject`. Questa struttura, inoltre, contiene le informazioni temporali dei vari modelli.

FCL è una libreria *C++* basata su più modelli in grado di eseguire diverse *query* di collisione e prossimità tra politopi convessi, modelli poligonali generali, modelli articolati, modelli deformabili e nuvole di punti. Inoltre, *FCL* è in grado di effettuare un controllo probabilistico delle collisioni in caso di nuvole di punti provenienti da fotocamere o sensori *LIDAR*.

Il nodo che funge da *API* per la verifica delle collisioni e per i calcoli di prossimità è il *Traversal Node*. Esso si occupa di effettuare le seguenti verifiche:

- Collisione discreta:

  Dati due oggetti A e B e le loro rispettive configurazioni $\boldsymbol{q_A}$ e $\boldsymbol{q_B}$ la *query* di collisione discreta (DCD) effettua un controllo del tipo:

  $$\boldsymbol{A}(\boldsymbol{q_A}) \cap \boldsymbol{B}(\boldsymbol{q_B}) \neq \boldsymbol{0} \qquad (3.13)$$

  Restituisce una variabile booleana:
  - ***true*** se la condizione è vera e i due corpi sono in collisione;
  - ***false*** se i due corpi non sono in collisione.

- Collisione continua:

  Dati due oggetti A e B e i rispettivi movimenti $\boldsymbol{q_A}(t)$ e $\boldsymbol{q_B}(t)$ con $t \in [0; t_f]$, la *query* di collisione continua (CCD) effettua un controllo del tipo:

  $$\exists\, t \in [0; t_f], \boldsymbol{A}(\boldsymbol{q_A}(t)) \cap \boldsymbol{B}(\boldsymbol{q_B}(t)) \neq \boldsymbol{0} \qquad (3.14)$$

  Restituisce una variabile booleana:
  - ***true*** se la condizione è vera e i due corpi sono in collisione;
  - ***false*** se i due corpi non sono in collisione.

  Se si verifica una collisione viene anche restituito il primo punto in contatto:

  $$toc = inf\ \{t : \boldsymbol{A}(\boldsymbol{q_A}(t)) \cap \boldsymbol{B}(\boldsymbol{q_B}(t)) \neq \boldsymbol{0}\} \qquad (3.15)$$

- Distanza di separazione:

  Dati due oggetti non sovrapposti A e B, nonché le loro configurazioni $\boldsymbol{q_A}$ e $\boldsymbol{q_B}$, la *query* di distanza di separazione (SD) effettua un calcolo sulla distanza che separa i due oggetti:

$$dis = inf\ \{||\ \boldsymbol{x} - \boldsymbol{y}\ ||_2 :\ \boldsymbol{x} \in \boldsymbol{A}(\boldsymbol{q_A})\ , \boldsymbol{y} \in \boldsymbol{B}(\boldsymbol{q_B})\} \qquad (3.16)$$

- Profondità di penetrazione:

  Quando invece, i due oggetti sono in collisione, viene calcolata la profondità di penetrazione attraverso la relazione:

$$pd = inf\ \{||\ \boldsymbol{d}\ ||_2 :\ (\boldsymbol{A}(\boldsymbol{q_A}) + \boldsymbol{d})\ \cap\ \boldsymbol{B}(\boldsymbol{q_B}) = \boldsymbol{0}\} \qquad (3.17)$$

Essendo questo calcolo molto laborioso dal punto di vista computazionale, l'algoritmo *FCL*, restituisce solo una stima della profondità di penetrazione tra due modelli basati su mesh.

Fonte ([27] - Jia Pan, Sachin Chitta, Dinesh Manocha -, 2012).

Oggetti di collisione

MoveIt supporta il controllo di collisione per diversi tipi di oggetti tra cui:

- *Meshes* – è possibile importare all'interno dell'ambiente di simulazione del robot file in formato *.stl* o *.dae*;
- Forme primitive – es. Cubi, Cilindri, Coni, Sfere e Piani;
- *Octomap* – L'oggetto *Octomap* può essere usato direttamente per il controllo di collisione.

Cinematica

La **cinematica diretta** è lo strumento che permette la trasformazione delle coordinate in spazio giunti a coordinate in spazio operativo. Al contrario, la **cinematica inversa** permette di trasformare le coordinate in spazio operativo a coordinate in spazio giunti. Infatti, come è ben noto dalla letteratura, una delle principali difficoltà che si incontra nello studio dei manipolatori è la ricerca di una soluzione per la conversione delle coordinate in spazio operativo a coordinate in spazio giunti.

La **cinematica diretta** è lo strumento che permette la trasformazione delle coordinate in spazio giunti a coordinate in spazio operativo attraverso la relazione:

$$\boldsymbol{x_e} = \boldsymbol{f}(\boldsymbol{q}) \qquad (3.18)$$

Con:

- $\boldsymbol{x_e}$ il vettore $(r \times 1)$ che contiene le informazioni delle coordinate dell'organo terminale nello spazio operativo;
- $\boldsymbol{q}$ il vettore $(n \times 1)$ che contiene le informazioni delle coordinate spazio giunti;
- $\boldsymbol{f}$ funzione vettoriale che permette la trasformazione delle coordinate in spazio giunti in coordinate in spazio operativo. La funzione vettoriale dipende dalla struttura del manipolatore.

Per quanto riguarda invece il calcolo delle velocità di traslazione e rotazione dell'organo terminale, conoscendo le velocità dei giunti che costituiscono il manipolatore, si fa riferimento alla cinematica differenziale e all'equazione:

$$\boldsymbol{v_e} = \boldsymbol{J}(\dot{\boldsymbol{q}}) \qquad (3.19)$$

Con:

- $\boldsymbol{v_e}$ il vettore $(r \times 1)$ che contiene le informazioni delle velocità dell'organo terminale in spazio operativo;
- $\dot{\boldsymbol{q}}$ il vettore $(n \times 1)$ che contiene le informazioni delle velocità in spazio giunti;
- $\boldsymbol{J}$ matrice Jacobiana dimensioni $(r \times n)$.

L'operazione di **inversione cinematica** permette di risalire al valore che bisogna imporre alle variabili di giunto per ottenere una specifica posa cartesiana da parte del manipolatore. La relazione che lega la posa del manipolatore nello spazio operativo e il vettore delle variabili di giunto è una relazione non lineare.

Un modo adottabile per risolvere il problema è avvalersi dell'equazione cinematica differenziale:

$$\boldsymbol{q} = \boldsymbol{J}^{+}(\boldsymbol{q})\boldsymbol{p} \quad (3.20)$$

Con:

- $\boldsymbol{p}$ il vettore posa del manipolatore;
- $\boldsymbol{J}^{+} = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1}$ è la pseudoinversa dello Jacobiano analitico;
- $\boldsymbol{q}$ il vettore delle configurazioni del robot nello spazio giunti.

Fonte ([19] - Stephen L. Chiu, 1988).

Plugin della cinematica

La cinematica diretta e la ricerca di Jacobiani sono integrati in *MoveIt* nella classe *RobotState*.

Il plug-in di cinematica inversa predefinito per *MoveIt* è *KDL*, un risolutore numerico che sfrutta la pseudoinversa dello Jacobiano per la risoluzione di problemi di cinematica inversa.

Fonte ([8] - MoveIt!, s.d.).

Per la nostra applicazione specifica si è integrato il risolutore di cinematica inversa *TRAC-IK*.

TRAC-IK è un risolutore di cinematica inversa sviluppato da *TRACLabs* che combina due algoritmi risolutori di cinematica inversa per ottenere soluzioni più affidabili rispetto ai comuni risolutori *open source* disponibili.

TRAC-IK esegue contemporaneamente due implementazioni di cinematica inversa:

1) Algoritmo ***KDL-RR***: che permette di rilevare e risolvere le problematiche di discontinuità dovute ai minimi locali in corrispondenza dei limiti di giunto;

2) Algoritmo ***SQP-SS***: che attraverso un algoritmo di programmazione quadratica sequenziale permette di gestire meglio le condizioni in prossimità dei limiti di giunto.

Non appena uno di questi due algoritmi converge ad una soluzione, *TRAC-IK* arresta la ricerca e restituisce immediatamente il risultato.

I metodi di risoluzione di cinematica inversa si avvalgono della pseudoinversa dello Jacobiano e risultano essere abbastanza eleganti nonostante la loro semplicità.

Avendo un vettore "seme" dei giunti $\boldsymbol{q_{seme}}$ (solitamente il valore dei giunti allo stato attuale), la cinematica diretta può essere utilizzata per calcolare:

1- La posa cartesiana nella configurazione $\boldsymbol{q_{seme}}$;

2- il vettore $\boldsymbol{p_{err}}$ appartenente ad R^6 in cui è definito l'errore cartesiano tra la "posa seme" e la "posa obiettivo";

3- lo Jacobiano $\boldsymbol{J}$ definisce le derivate parziali nello spazio cartesiano rispetto ai valori di giunto attuali. Dopo aver invertito lo Jacobiano $\boldsymbol{J^{-1}}$ vengono definite le derivate parziali nello spazio giunti rispetto allo spazio cartesiano.

Di conseguenza, la soluzione di cinematica inversa viene semplicemente calcolata iterando la funzione:

$$\boldsymbol{q_{successivo}} = \boldsymbol{q_{precedente}} + \boldsymbol{J^{+}p_{err}} \qquad (3.21)$$

Algoritmo di inversione cinematica *KDL-RR*

L'algoritmo *KDL-RR* è un risolutore di cinematica inversa basato sul precedente risolutore *KDL*. *KDL* richiede in input un numero massimo di iterazioni da tentare durante la ricerca per trovare una soluzione di cinematica inversa, mentre, *KDL-RR* richiede in input una finestra temporale entro cui deve effettuare la ricerca di una soluzione.

KDL è un risolutore di cinematica inversa che sfrutta un algoritmo ad anello chiuso basato sullo Jacobiano pseudoinverso.

KDL può arrestare la sua ricerca in prossimità dei minimi locali durante il processo di iterazione (in prossimità dei limiti di giunto). In *KDL* non c'è nessuno strumento per rilevare i minimi locali oppure per superare le difficoltà del calcolatore in presenza di limiti di giunto fisici.

KDL-RR rileva i minimi locali tramite la semplice operazione:

$$\boldsymbol{q_{successivo}} - \boldsymbol{q_{precedente}} = \boldsymbol{0} \qquad (3.22)$$

In definitiva, in *KDL-RR,* viene implementato un risolutore di cinematica inversa che sfrutta anche in questo caso lo Jacobiano pseudoinverso, ma in questo caso vengono rilevati anche i minimi locali e per evitare il blocco in prossimità di tali configurazioni, viene cambiato il seme $\boldsymbol{q_{seme}}$ passando ad uno successivo. Le prestazioni del risolutore *KDL* di cinematica inversa vengono notevolmente migliorate semplicemente utilizzando un seme random $\boldsymbol{q}$ per sbloccare lo stallo dell'algoritmo iterativo quando viene riscontrato un minimo locale.

L'implementazione di *KDL* con riavvii casuali e basato su un tempo limitato è noto come *KDL-RR*.

Algoritmo di inversione cinematica *SPQ-SS*

L'analisi dei fallimenti in *KDL-RR* ha mostrato che molti errori si verificano quando l'algoritmo iterativo sembra fare progressi tali che $\Delta\boldsymbol{q} \neq \boldsymbol{0}$ ma la configurazione del robot sta andando verso limiti di giunto. Nelle applicazioni teoriche, dove i giunti sono illimitati, gli algoritmi che sfruttano lo Jacobiano inverso funzionano abbastanza bene; però, in pratica, molti giunti robotici hanno limiti fisici.

Il risolutore di cinematica inversa sfrutta un algoritmo iterativo per l'ottimizzazione non lineare. La cinematica inversa viene risolta localmente risolvendo problemi di ottimizzazione non lineare utilizzando una programmazione quadratica sequenziale (*SPQ*).

L'algoritmo di riferimento prevede la risoluzione del seguente problema:

$$arg \min_{\boldsymbol{q} \in R^n} (\boldsymbol{q_{seme}} - \boldsymbol{q})^T (\boldsymbol{q_{seme}} - \boldsymbol{q}) \qquad (3.23)$$

$$C.C.: \quad f_i(\boldsymbol{q}) \leq b_i, i = 1, \dots, m.$$

Con:

- $\boldsymbol{q_{seme}}$ il vettore n-dimensionale dei giunti;
- Le condizioni al contorno $f_i(\boldsymbol{q})$ possono essere i limiti di giunto, l'errore di distanza Euclidea e l'errore di distanza angolare.

La funzione obiettivo per minimizzare l'errore cartesiano in *SPQ-SS* utilizza la somma dei quadrati:

$$\Phi_{err} = \boldsymbol{p_{err}} - \boldsymbol{p_{err}^T} \qquad (3.24)$$

Fonte ([14] - Patrick Beeson and Barrett Ames, , 2015).

All'interno della sezione dedicata ai risolutori di cinematica inversa è possibile specificare il criterio di risoluzione tra i seguenti:

- `solve_type: Speed`. Seguendo questo criterio, *TRAC-IK* restituisce rapidamente la prima soluzione trovata;
- `solve_type: Distance`. Seguendo questo criterio, *TRAC-IK* esegue iterativamente una serie di istruzioni per tutto il tempo computazionale imposto e al termine, restituisce la soluzione che minimizza la distanza rispetto al nodo seme;
- `solve_type: Manipulation1`. Seguendo questo criterio, vengono eseguite iterativamente una serie di istruzioni per tutto il tempo computazionale imposto e al termine *TRAC-IK* restituisce la soluzione che massimizza l'**indice di manipolabilità**;
- `solve_type: Manipulation2`. Seguendo questo criterio, vengono eseguite iterativamente una serie di istruzioni per tutto il tempo computazionale imposto e al termine *TRAC-IK* restituisce la soluzione che minimizza il **numero di condizionamento**;

Riferimento ([17] - TRACLABS, s.d.).

L'**indice di manipolabilità** viene definito come $w = sqrt(det\ (\boldsymbol{J}(\theta)\boldsymbol{*J}^T(\theta))$ in caso di manipolatori ridondanti. La manipolabilità misura l'attitudine del manipolatore a trasmettere velocità all'organo terminale. Valori prossimi allo zero o troppo elevati individuano zone in cui il manipolatore non lavora in condizioni ottimali (es. in prossimità della frontiera dello spazio di lavoro). Fonte ([16] - Tsuneo Yoshikawa, 1985)

Il **numero di condizionamento** fornisce una valutazione locale della destrezza del manipolatore, in quanto dipende dalla configurazione dello stesso. Il numero di condizionamento della matrice Jacobiana [$\boldsymbol{J}$] è espresso come segue:

$$K(\boldsymbol{J}) = sqrt(\frac{\lambda min}{\lambda max}) \qquad (3.25)$$

λmin = autovalore min della matrice $\boldsymbol{J}$;

λmax = autovalore max della matrice $\boldsymbol{J}$.

Oppure anche:

$$c(\boldsymbol{J^T}) = ||\boldsymbol{J^T}|| * ||\boldsymbol{J^{-T}}|| \qquad (3.26)$$

Con $||\ \cdot\ ||$ Norma della matrice.

I punti dello spazio di lavoro che minimizzano il numero di condizionamento della matrice Jacobiana sono i punti meglio condizionati per minimizzare l'errore di propagazione dalle coppie in ingresso e le forze in uscita. I punti meglio condizionati sono quelli con numero di condizionamento $c(\boldsymbol{J^T}) = 1$.

Fonte ([18] - J. Kenneth Salisbury, John J. Craig, 1982).

4. Analisi Comparativa algoritmi *OMPL*

Il prototipo *Agriq.02* è stato pensato per svolgere semplici compiti di campionamento come, ad esempio, la raccolta di un oggetto dal suolo oppure, la raccolta di un grappolo d'uva da una vigna.

Di seguito due immagini che immortalano il robot immerso in ambiente di simulazione, dedito alla raccolta di un oggetto dal suolo e alla raccolta di un grappolo d'uva da una vigna.

Per simulare in maniera ottimale l'ambiente entro cui il rover dovrà operare, sono stati inseriti all'interno dell'ambiente di simulazione il modello 3D di *Agri.q02*, il modello 3D di un grappolo d'uva e delle paratie rettangolari laterali distanziate di 2 m, che stanno a rappresentare i filari di un vigneto. Per quanto riguarda la raccolta dell'oggetto, è stato inserito un cubo sul suolo.

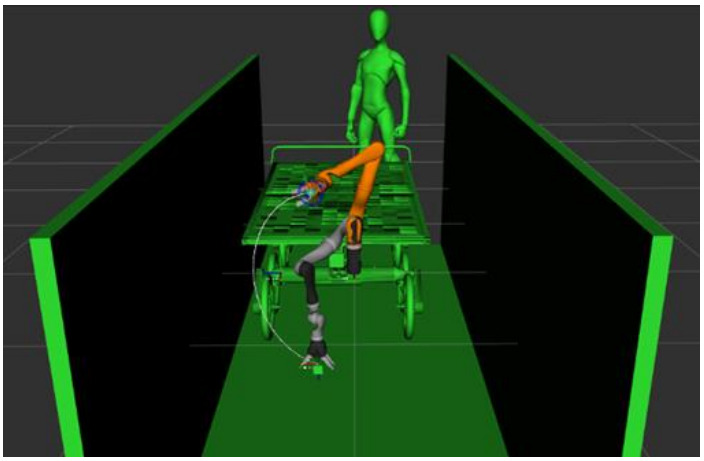
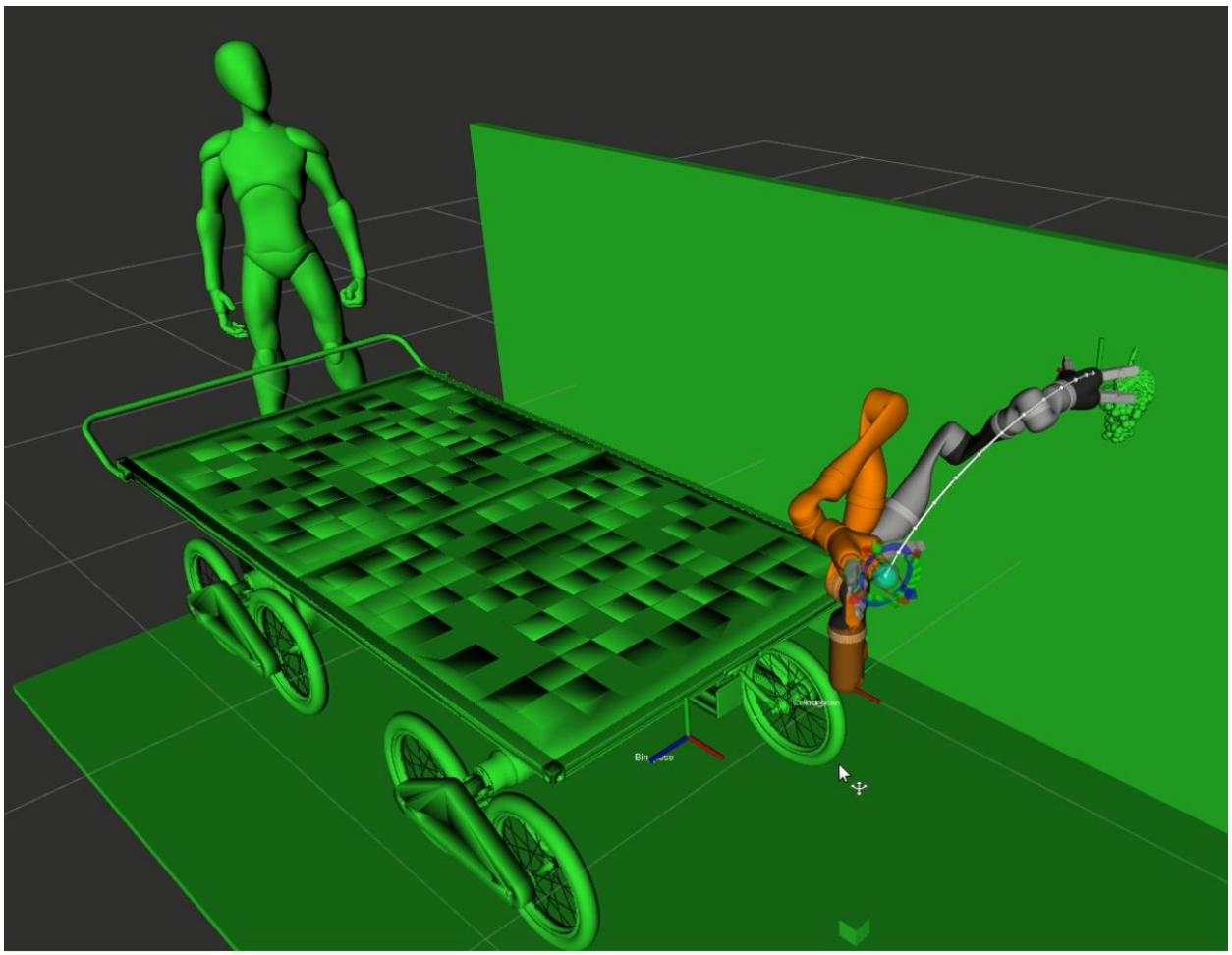


Figura 38 - Modello 3D dell'ambiente all'interno del quale il robot dovrà eseguire le operazioni. In alto è riportata un'immagine raffigurante il braccio robotico dedito alla raccolta di un oggetto dal suolo. In basso il robot è immortalato nel raccogliere un grappolo d'uva. In arancione, la configurazione di partenza del manipolatore, mentre, in grigio, la posizione di raccolta oggetto. La linea bianca che collega il TCP del manipolatore arancione e il TCP del manipolatore grigio, è la traiettoria eseguita per spostarsi da una configurazione all'altra.



Il software di visualizzazione tridimensionale impiegato per l'esecuzione delle prove è stato ***rviz***.

Attraverso l'utilizzo del linguaggio di programmazione *C++* è stato possibile controllare il manipolatore per la pianificazione e l'esecuzione di traiettorie, nonché, per operazioni di raccolta e rilascio di oggetti. Si rimanda alle fonti ([24] - MoveIt ROS Planning Interface, s.d.) e ([25] - MoveIt Visual Tools, s.d.) per le informazioni relative alle righe di codice utilizzate per controllare il manipolatore e per visualizzare le traiettorie nell'ambiente di simulazione.

Si è quindi realizzato un codice in grado di simulare l'ambiente di lavoro all'interno del quale il robot dovrà lavorare. I modelli 3D inseriti all'interno dell'ambiente di lavoro sono considerati degli ostacoli. Pertanto, il nodo `move_group` in fase di pianificazione delle traiettorie dovrà tenere conto di questi aspetti in modo da non far collidere nessun componente del manipolatore con i seguenti oggetti:

- Agriq.02;
- le due paratie laterali;
- il grappolo d'uva;
- l'oggetto sul suolo;
- pavimento;

Si fa presente che il manichino essendo fuori dall'area di lavoro del manipolatore non verrà verificato in fase di pianificazione di nessuna traiettoria.

La richiesta di pianificazione della traiettoria per la raccolta di un oggetto dal suolo e la successiva deposizione di tale oggetto in una determinata posizione richiede, in generale, poche problematiche. Generalmente, sono necessari pochi decimi di secondo per la pianificazione di semplici traiettorie, mentre, per la pianificazione di traiettorie con vincoli di orientamento, si sono riscontrate non poche difficoltà che verranno descritte in seguito.

4.1. Pianificazione di traiettorie con vincoli di orientamento

Per l'esecuzione della raccolta del grappolo d'uva si è imposto un vincolo di orientamento all'organo terminale del manipolatore, in modo che quest'ultimo, in fase di movimentazione del grappolo, non presentasse rotazioni attorno agli assi X e Z riferiti al SR nel *Tool Center Point* (d'ora in poi TCP). Riferimento Figura 39.

Orientamento dei SR nel TCP e alla base del manipolatore:

Figura 39 - Nell'immagine sulla sinistra è possibile mostrare il S.R. centrato nel TCP. Sulla destra invece è riportato il S.R. centrato alla base del manipolatore.

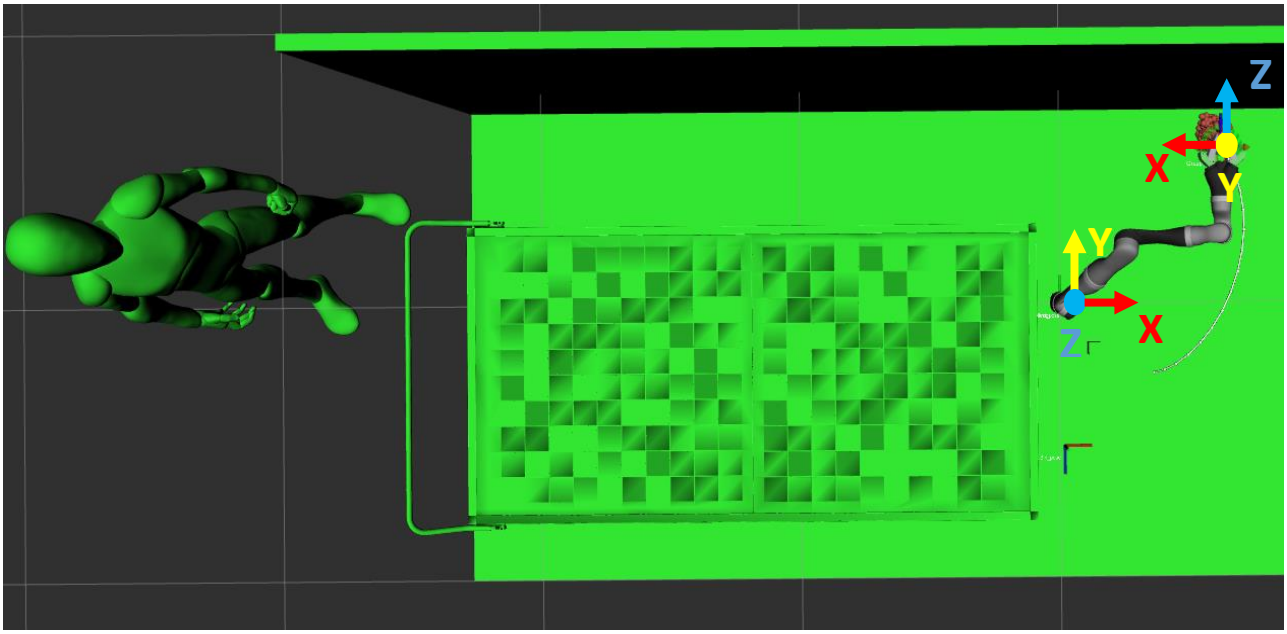


*Figura 40 – Vista dall'alto dell'ambiente di simulazione in **rviz**.*

Si fa presente che all'interno di *MoveIt*, la pianificazione del percorso con vincoli di orientamento avviene in spazio operativo, e, il risolutore di cinematica inversa (*TRAC-IK*) funge da "campionatore generativo". Fonte ([15] - MoveIt, s.d.) ***OMPL Settings → Enforce Planning in Joint Space.***

Il risultato di questa pianificazione è un manipolatore che esegue movimenti imprevedibili ed indefiniti. Infatti, in ambiente di simulazione è possibile notare come il manipolatore non segua la traiettoria pianificata, andando a collidere con gli oggetti che ha attorno, compenetrando la materia.

La natura di tale fenomeno risiede nel risultato della pianificazione della traiettoria: a differenza di quanto avviene in fase di pianificazione del percorso in spazio giunti, il risolutore di cinematica inversa, campiona pochi punti di via per l'esecuzione della traiettoria.

A titolo di esempio, nell'immagine riportata di seguito, sono raffigurati i comportamenti dei segnali di *set* e *feedback* di velocità del giunto 2, nell'esecuzione di una traiettoria con vincoli di orientamento generata dal risolutore di cinematica inversa.

Legenda:

- $\dot{q}_2$: *set* di velocità imposti al giunto 2 per l'esecuzione della traiettoria;
- $\dot{q}_2$: interpolazione lineare dei *set* di velocità imposti al giunto 2;
- $\dot{q}_2$: *feedback* di velocità giunto 2.

1 I segnali di *set* e *feedback* registrati all'interno del rettangolo tratteggiato in colore verde, raffigurano l'esecuzione del moto pianificato dall'adattatore *"FixStartStatePathConstraints"*. Come precedentemente accennato nel paragrafo "*FixStartStatePathConstraints* - Correttore dei vincoli di percorso dello stato di avvio", questo adattatore viene richiamato quando lo stato iniziale del manipolatore non rispetta i vincoli di orientamento richiesti, e pianifica una traiettoria per portare il manipolatore dalla configurazione iniziale ad una nuova configurazione nella quale viene rispettato il vincolo di orientamento imposto. La nuova configurazione, dunque, servirà come stato iniziale per la pianificazione con vincolo di orientamento. Tuttavia, durante tale fase preparatoria, non è assicurato il rispetto del vincolo di orientamento richiesto.

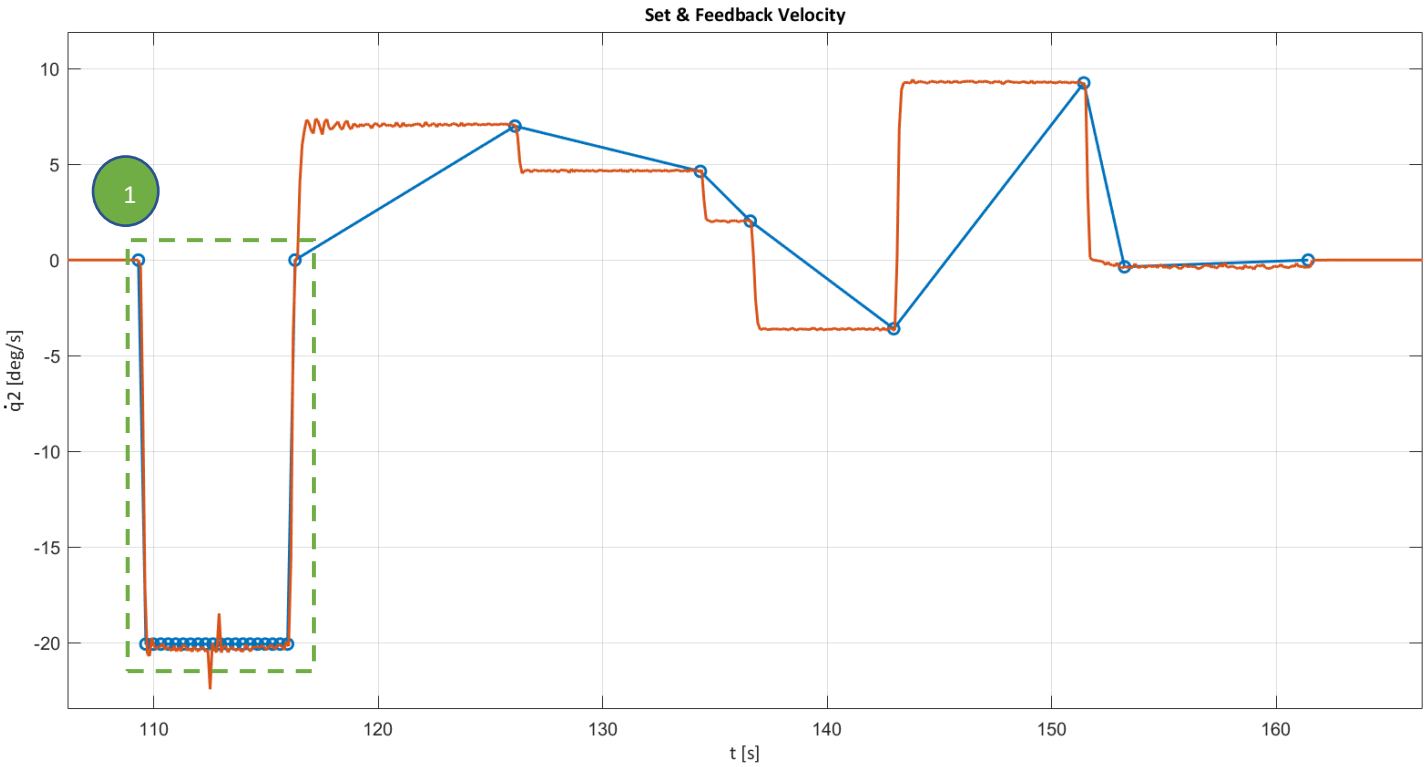


Figura 41 - Grafico in cui sono stati riportati i segnali di set e feedback di velocità imposti al giunto 2 per l'esecuzione di una traiettoria con vincoli di orientamento. TRAC-IK campionatore generativo.

Successivamente a tale fase preparatoria, inoltre, l'esecuzione della traiettoria (segnale di *feedback*) non segue la retta spezzata che interpola i punti celesti (segnali di *set* di velocità), come è possibile apprezzare dal grafico in Figura 41.

Questo comportamento è dovuto al comando fornito ai driver del manipolatore all'interno del file `kinova-ros/kinova_driver/src/kinova_joint_trajectory_controller.cpp`. Di seguito sono riportate le righe di codice che spiegano tale comportamento, le quali vengono di seguito commentate:

```
void JointTrajectoryController::pub_joint_vel(const ros::TimerEvent&)
{
    // send out each velocity command with corresponding duration delay.

    kinova_msgs::JointVelocity joint_velocity_msg;

    if (traj_command_points_index_ <  kinova_angle_command_.size() && ros::ok())

    {
        joint_velocity_msg.joint1 = kinova_angle_command_[traj_command_points_index_].Actuator1;
        joint_velocity_msg.joint2 = kinova_angle_command_[traj_command_points_index_].Actuator2;
        joint_velocity_msg.joint3 = kinova_angle_command_[traj_command_points_index_].Actuator3;
        joint_velocity_msg.joint4 = kinova_angle_command_[traj_command_points_index_].Actuator4;
        joint_velocity_msg.joint5 = kinova_angle_command_[traj_command_points_index_].Actuator5;
        joint_velocity_msg.joint6 = kinova_angle_command_[traj_command_points_index_].Actuator6;
        joint_velocity_msg.joint7 = kinova_angle_command_[traj_command_points_index_].Actuator7;

        pub_joint_velocity_.publish(joint_velocity_msg);
```

```
if( (ros::Time::now() - time_pub_joint_vel_) >= traj_command_points_[traj_command_points_index_].time_from_start)
    {
        ROS_INFO_STREAM("Moved to point " << traj_command_points_index_++);
    }
}
```

Come nell'esecuzione di tutte le traiettorie, per ciascun giunto, vi è l'imposizione di un *set* di velocità strettamente legato ad un istante temporale.

`kinova_angle_command_[traj_command_points_index_].Actuator2`

`traj_command_points_[traj_command_points_index_].time_from_start`

$\dot{q}_1$	$\dot{q}_2$	…	…	…	…	$\dot{q}_n$
t_1	t_2	…	…	…	…	t_n

Con $n =$ `traj_command_points_.size()`, il numero totale dei valori di *set* da imporre ai giunti per l'esecuzione di un determinato movimento.

Le caselle dei vettori contenenti le informazioni dei *set* di velocità e le informazioni temporali della traiettoria, vengono identificate da un indice: `traj_command_points_index_`.

Il vettore `traj_command_points_[].time_from_start` contiene le informazioni temporali relative ai *set* di velocità da imporre a ciascun giunto, a partire dall'inizio dell'esecuzione della traiettoria. Tra due intervalli temporali $t_{i-1} < t < t_i$ viene imposto un *set* di velocità pari a $\dot{q}_i$.

Il vettore `kinova_angle_command_[].Actuator2` contiene le informazioni di *set* di velocità del giunto 2.

Per ogni traiettoria, viene registrato l'istante di tempo in cui la sua esecuzione inizia, salvando questo valore all'interno del parametro `time_pub_joint_vel_` .

Durante il movimento, nell'esecuzione di ciascun tratto del percorso, vi è un controllo in tempo reale:

```
(ros::Time::now() - time_pub_joint_vel_) >= traj_command_points_[traj_command_points_index_].time_from_start
```

Per cui viene verificato il tempo intercorso dall'inizio dell'esecuzione della traiettoria, fino all'istante attuale. Se tale differenza risulta maggiore o uguale t_i , il driver fornirà dei *set* di velocità ai motori dei singoli giunti, pari a $\dot{q}_{i+1}$, ovvero, il valore successivo presente all'interno del vettore `kinova_angle_command_[].Actuator2`.

Questo metodo risulta efficace fintantoché i segnali di *set* di velocità presentano una discretizzazione temporale dell'ordine dei decimi di secondo, poiché, il manipolatore segue in maniera più o meno affidabile la traiettoria pianificata.

Nel grafico mostrato in precedenza, l'intervallo temporale che intercorre tra due valori successivi t_{i-1} e t_i è di circa 10 secondi. Durante tale intervallo, l'*API* dei driver, impone un *set* di velocità costante pari a $\dot{q}_i$. Il giunto 2, infatti, non segue in alcun modo l'andamento lineare della retta spezzata che congiunge i punti celesti di *set* di velocità, ma, anzi, presenta una legge di velocità molto simile a una trapezoidale.

In queste condizioni, la pianificazione della traiettoria risulta poco efficace e addirittura potenzialmente dannosa per l'incolumità del manipolatore stesso e di tutti gli oggetti che lo circondano.

Di seguito è riportata un'immagine del manipolatore che ha completato il suo movimento trovandosi in una configurazione indefinita.

Legenda:

- Manipolatore reale che ha terminato il movimento;
- Configurazione che avrebbe dovuto raggiungere il manipolatore una volta terminato il movimento.

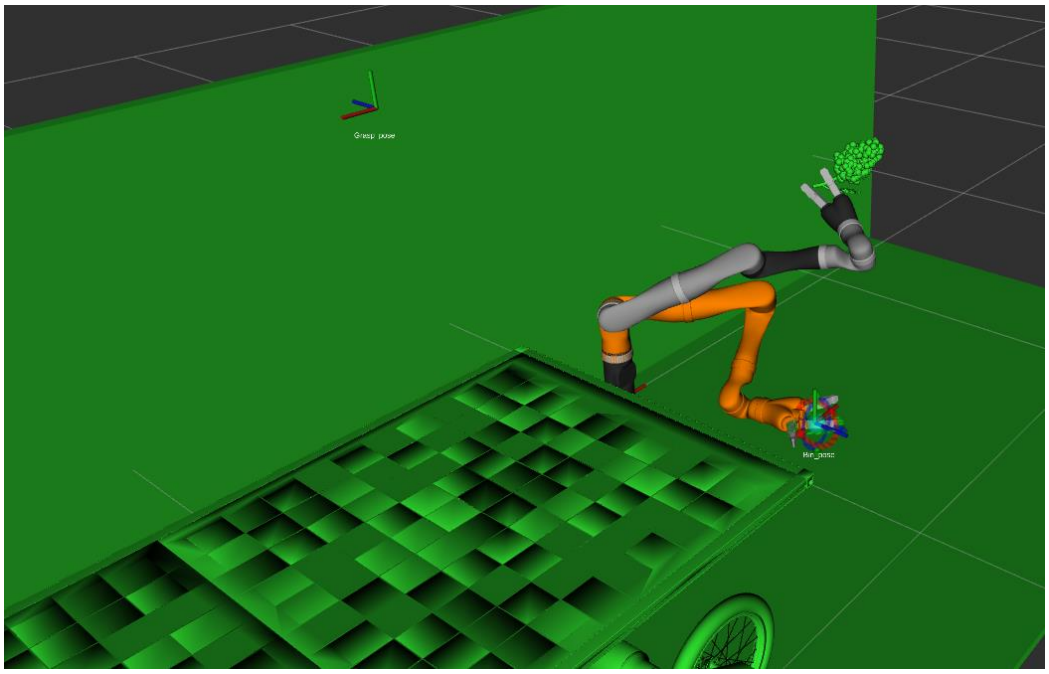


Figura 42 - Il manipolatore di colore arancione rappresenta la configurazione che avrebbe dovuto raggiungere al termine del movimento per il rilascio del grappolo d'uva. Il manipolatore di colore grigio, invece, rappresenta il manipolatore che ha terminato l'esecuzione della traiettoria.

Inoltre, nell'esecuzione delle traiettorie, il manipolatore, ha raggiunto in diverse occasioni la saturazione dei limiti di giunto. Nell'immagine sottostante vi è un esempio del raggiungimento dei limiti di giunto nell'esecuzione di una traiettoria per i giunti 4 e 6:

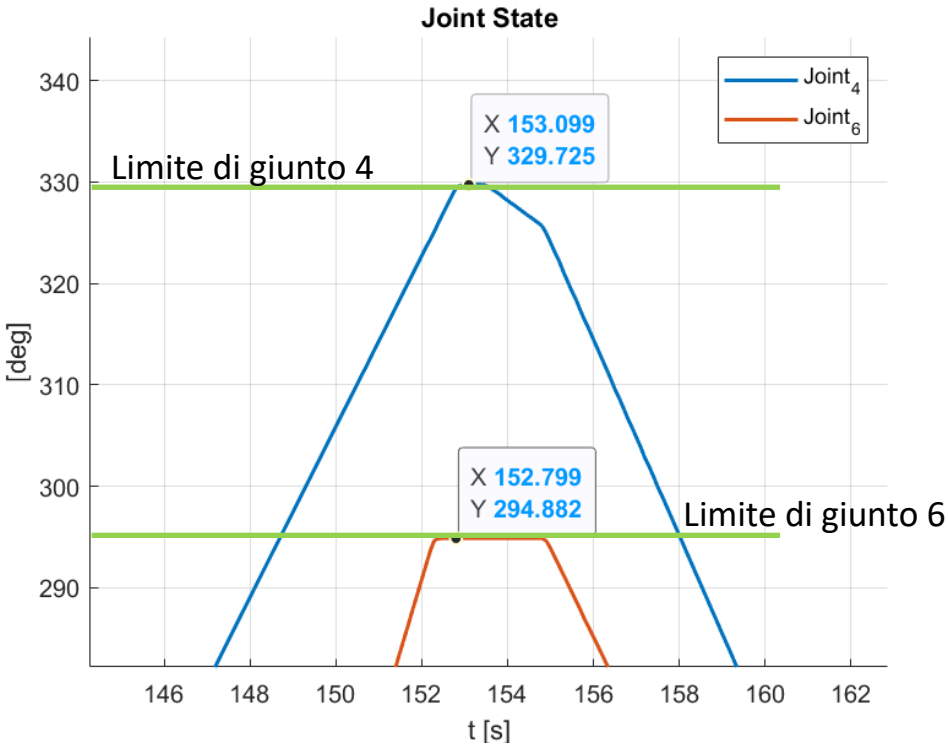


Figura 43 - Grafico dello stato giunti durante l'esecuzione di una traiettoria. Il manipolatore è giunto a saturazione per i giunti 4 e 6 rendendo impossibile l'esecuzione del percorso tracciato.

Dato questo effetto indesiderato, in quanto pericoloso sia per *Agri.q02*, sia per l'incolumità di un operatore che potrebbe trovarsi nell'area di lavoro del manipolatore, si è vincolata la pianificazione di percorsi in spazio giunti.

Si rimanda all'**Appendice – B** per ulteriori informazioni su come imporre la pianificazione del percorso in spazio giunti.

4.2. Test Sperimentali

Avendo raggiunto una piena consapevolezza degli strumenti utilizzati, si è deciso di verificare quale pianificatore di percorsi meglio si adattasse alla specifica applicazione.

È stato quindi sviluppato un algoritmo di test affinché si potesse valutare il miglior pianificatore in grado di ricostruire una traiettoria tale da evitare gli ostacoli e da rispettare il vincolo di orientamento imposto.

Per questa ragione si sono testati i pianificatori del moto nell'eseguire i seguenti compiti in successione:

- Posizione di Partenza $\boldsymbol{q} = [4.9\ 2.9\ 0.0\ 0.8\ 4.6\ 4.5\ 5.0]\ rad$;
- Posizione di raccolta grappolo;
- Posizione di rilascio grappolo all'interno di un eventuale cesto da equipaggiare su *Agri.q02*.

Durante questa analisi i parametri che sono stati tenuti in considerazione sono stati:

- L'indice di successo di pianificazione definito come: $\frac{\#\text{pianificazioni avvenute con successo}}{\text{tentativi di pianificazione totali}}$;
- Tempo computazionale per la pianificazione del moto.

Di seguito un'immagine volta a raffigurare la procedura logica di esecuzione degli esperimenti:

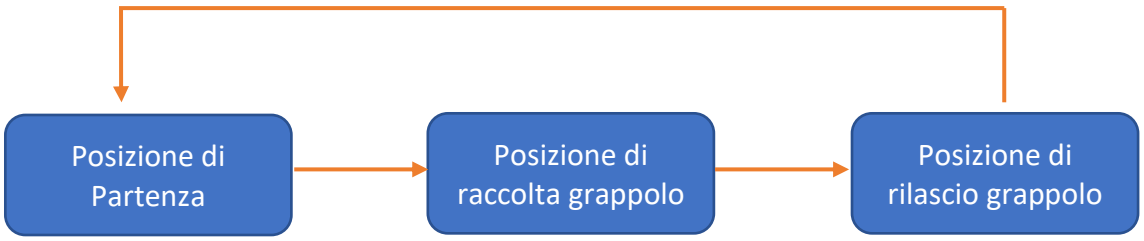


Figura 44 – Diagramma di flusso in cui sono riportati i movimenti fatti fare al robot per ciascuna prova.

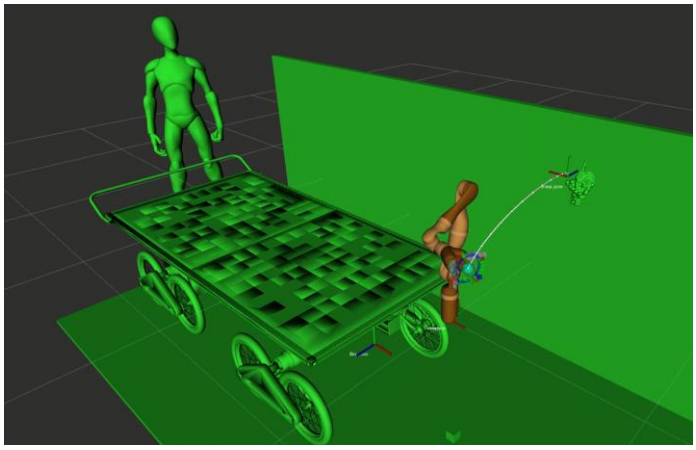
Dalla posizione di partenza (con le pinze aperte) viene inviata una richiesta di pianificazione di traiettoria al nodo `move_group` per il raggiungimento del punto di raccolta grappolo. Tale percorso è privo di vincoli di orientamento, pertanto, la pianificazione di questa traiettoria, solitamente, richiede pochi decimi di secondo. Se la richiesta di pianificazione va a buon fine, allora è possibile eseguire tale traiettoria, in caso contrario verrà inviata una nuova richiesta di pianificazione al nodo `move_group`.

Una volta che il manipolatore è in posizione di raccolta grappolo, vengono chiuse le pinze e il grappolo d'uva viene reso solidale all'organo terminale del manipolatore. A questo punto viene imposto il vincolo di orientamento all'organo terminale (vedi paragrafo 4.1) e viene inviata la richiesta di pianificazione al nodo `move_group`, per il raggiungimento della posizione di rilascio grappolo. La richiesta di tale operazione è onerosa dal punto di vista computazionale e pertanto richiede molto più tempo. I tempi per tale richiesta sono variabili funzione del pianificatore scelto.

Se la richiesta di pianificazione va a buon fine, allora, è possibile eseguire tale traiettoria, in caso contrario, verrà inviata una nuova richiesta di pianificazione. Come precedentemente accennato nel paragrafo "*FixStartStatePathConstraints* - Correttore dei vincoli di percorso dello stato di avvio", per far sì che l'esecuzione della traiettoria avvenga con vincolo di orientamento, viene richiamato l'adattatore "*FixStartStatePathConstrain",* al fine di pianificare una traiettoria iniziale, per portare il manipolatore dalla configurazione iniziale ad una nuova configurazione nella quale viene rispettato il vincolo di orientamento richiesto. La nuova configurazione servirà come stato iniziale per la pianificazione del percorso con vincolo di orientamento. Durante l'esecuzione di tale traiettoria, però, non è assicurato il rispetto del vincolo di orientamento richiesto.

Al termine dell'esecuzione della seconda traiettoria, viene inviata l'ultima richiesta di pianificazione al nodo `move_group`, per il raggiungimento della posizione di partenza. Anche in questo caso, se la richiesta di pianificazione va a buon fine, allora è possibile eseguire tale traiettoria, in caso contrario verrà inviata una nuova richiesta di pianificazione.

Posizione di partenza:



Posizione di raccolta grappolo:

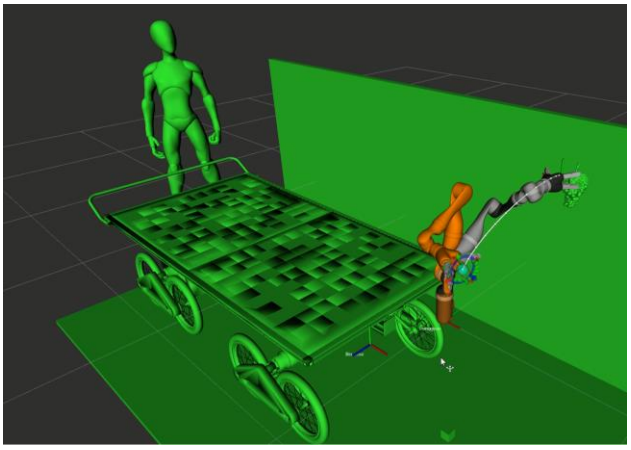


Figura 45 - Sulla sinistra è raffigurato l'ambiente di simulazione in cui il manipolatore è immerso. Il manipolatore è in "Posizione di partenza". Sulla destra il manipolatore reale è di colore grigio ed è in "Posizione di raccolta grappolo" mentre in arancione il manipolatore è nella posizione precedente ("Posizione di partenza"). In bianco è possibile apprezzare il percorso che l'organo terminale del manipolatore ha seguito per passare dalla configurazione arancione a quella grigia.

Posizione di rilascio grappolo:

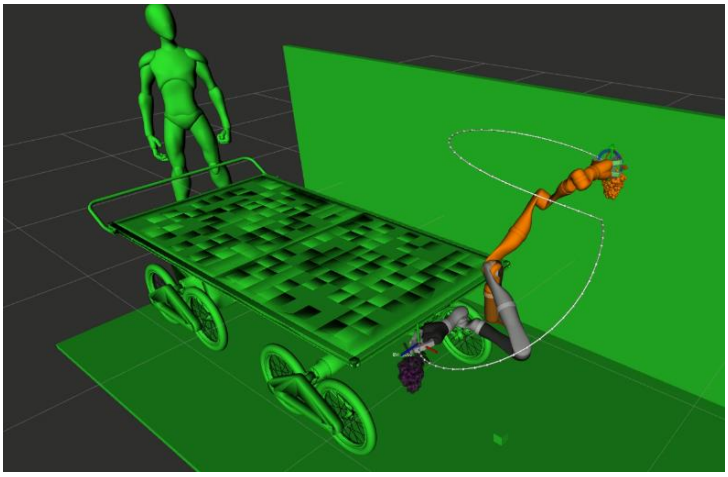


Figura 46 - Il manipolatore arancione è in "Posizione di raccolta grappolo" e il manipolatore reale di colore grigio in "Posizione di rilascio grappolo". In bianco è possibile apprezzare il percorso che l'organo terminale del manipolatore ha seguito per passare dalla configurazione arancione a quella grigia.

Per questo lavoro di tesi i pianificatori del moto che sono stati analizzati sono stati:

- *PRM;*
- *PRMStar;*

- *RRT;*
- *RRTConnect;*
- *RRT* (RRTStar).*

Test e risultati

Sono state condotte delle prove di raccolta grappolo in un intervallo di 0.90 m lungo l'asse X.

Mantenendo costante la profondità Y pari a 0.5 m e l'altezza Z pari a 0.7 m rispetto alla base del manipolatore (circa 1.3 m rispetto al suolo).

Avendo fatto variare la quota X rispetto alla base del manipolatore, si sono scelti 5 punti di raccolta ovvero: X = -0.45 m, -0.30 m, 0.0 m, 0.30 m e 0.45 m. Il manipolatore è stato in grado di raggiungere le suddette quote in configurazioni "agevoli" alla manipolazione. Il robot, infatti, non risultava "sbracciato" (in totale estensione) o in prossimità di configurazioni singolari.

Durante le prove condotte si sono imposti i seguenti parametri:

- Risolutore di cinematica inversa → `Track-IK solve_type: Manipulation2;`
- Parametrizzazione nel tempo → `Iterative Parabolic Time Parameterization (IPTP)`
- 10 tentativi di pianificazione della traiettoria per ciascun pianificatore.

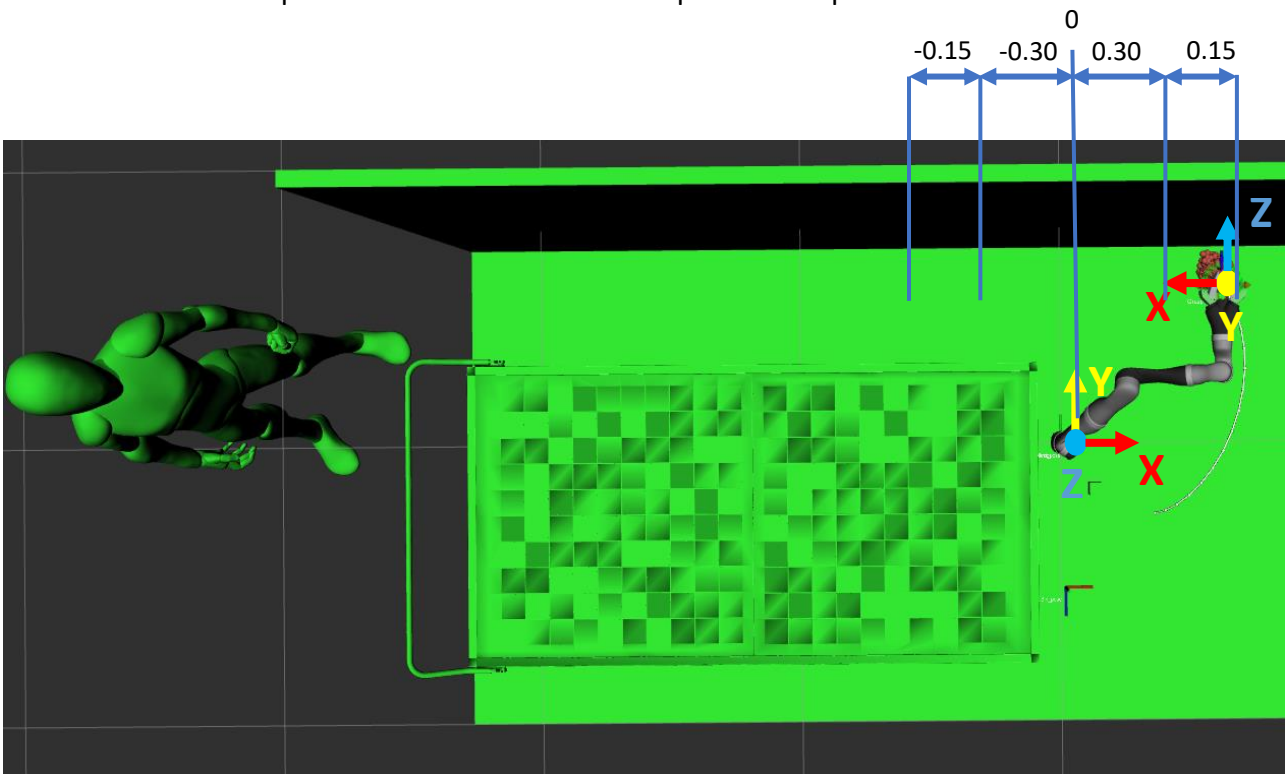


Figura 47 - Vista superiore dell'ambiente di lavoro del manipolatore. Sono indicati i cinque di punti raccolta grappolo per l'esecuzione dei test sperimentali.

Per i pianificatori *RRT*, *RRTConnect* ed *RRTstar* sono stati concessi 25 secondi per la pianificazione del percorso.

Per i pianificatori *PRM* e *PRMstar* sono stati concessi 15 secondi per la pianificazione del percorso.

Specifiche dispositivo:

Processore: *Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2.59 GHz*
RAM installata: *8,00 GB*
Scheda grafica: *NVIDIA GeForce GTX 950M*

Prova – 1

γ (angolo di beccheggio) =0	X [m]	Y [m]	Z [m]
Distanza tra i due filari [m]		2	
Raccolta grappolo SR Root (base del robot)	0	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15

Tabella 5 - Dati raccolta e rilascio grappolo Prova – 1.

I risultati dei pianificatori *RRT* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
RRT	Raccolta	10	100%	0,0316208
	Rilascio	10	70%	12,94458757
RRT Connect		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,0297647
	Rilascio	10	60%	17,209624
RRT star		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,1012115
	Rilascio	10	40%	25,011997

Tabella 6 – Risultati Prova – 1 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori RRT, RRTConnect e RRTstar.

I risultati dei pianificatori *PRM* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
PRM	Raccolta	10	100%	0,1086788
	Rilascio	10	90%	15,02678767
PRMstar		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,1969619
	Rilascio	10	60%	15,02532822

Tabella 7 – Risultati Prova – 1 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori PRM e PRMstar.

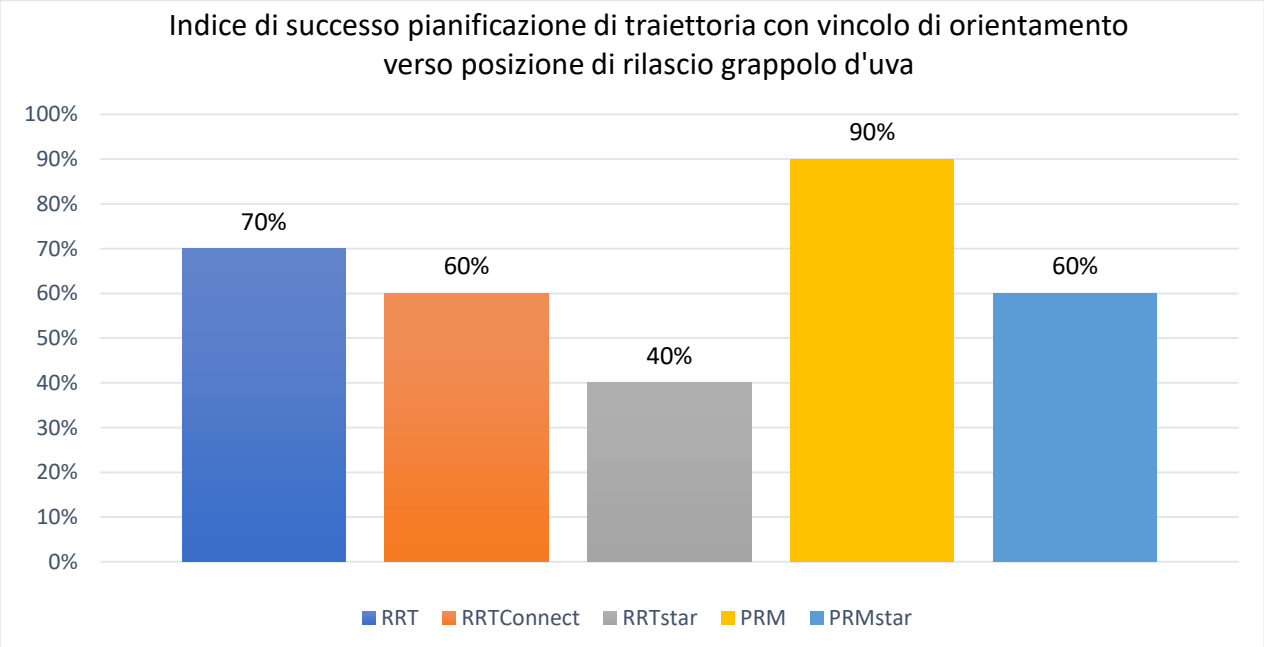


Figura 48 – Sintesi dei tentativi di pianificazione andati a buon fine per la Prova – 1 dei diversi pianificatori.

Prova – 2

γ (angolo di beccheggio) =0	X [m]	Y [m]	Z [m]
Distanza tra i due filari [m]		2	
Raccolta grappolo SR Root (base del robot)	0,30	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15

Tabella 8 - Dati raccolta e rilascio grappolo Prova – 2.

I risultati dei pianificatori *RRT* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
RRT	Raccolta	10	100%	0,0315174
	Rilascio	10	60%	20,39070733
RRT Connect		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,0301886
	Rilascio	10	100%	12,9333909
RRT star		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,0809835
	Rilascio	10	60%	25,00493267

Tabella 9 - Risultati Prova – 2 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori RRT, RRTConnect e RRTstar.

I risultati dei pianificatori *PRM* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
PRM	Raccolta	10	100%	0,0858968
	Rilascio	10	90%	15,01747744
PRMstar		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,096636
	Rilascio	10	100%	15,0515311

Tabella 10 - Risultati Prova – 2 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori PRM e PRMstar.

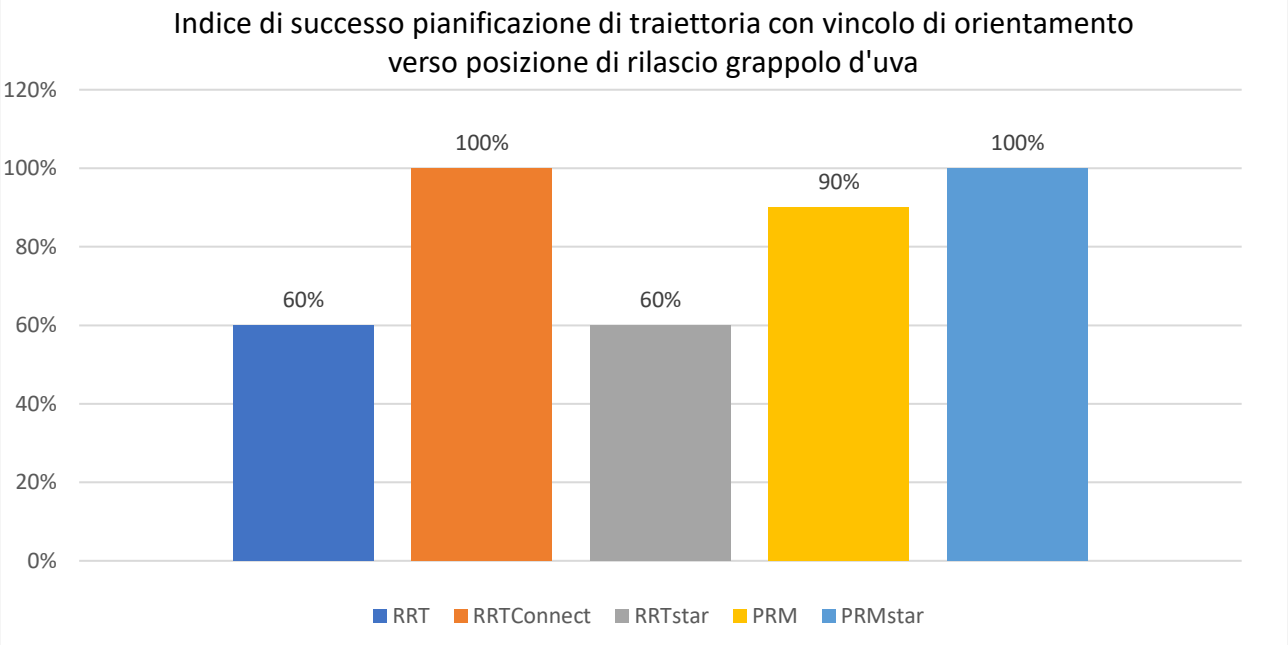


Figura 49 - Sintesi dei tentativi di pianificazione andati a buon fine per la Prova – 2 dei diversi pianificatori.

Prova – 3

γ (angolo di beccheggio) =0	X [m]	Y [m]	Z [m]
Distanza tra i due filari [m]		2	
Raccolta grappolo SR Root (base del robot)	-0,30	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15

Tabella 11 - Dati raccolta e rilascio grappolo Prova – 3.

I risultati dei pianificatori *RRT* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
RRT	Raccolta	10	100%	0,038458
	Rilascio	10	80%	14,364
RRT Connect		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,026583
	Rilascio	10	90%	14,21745
RRT star		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,101253
	Rilascio	10	90%	25,01811

Tabella 12 - Risultati Prova –3 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori RRT, RRTConnect e RRTstar.

I risultati dei pianificatori *PRM* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
PRM	Raccolta	10	100%	0,108294
	Rilascio	10	80%	15,02823
PRMstar		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,106565
	Rilascio	10	90%	15,04077

Tabella 13 - Risultati Prova – 3 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori PRM e PRMstar.

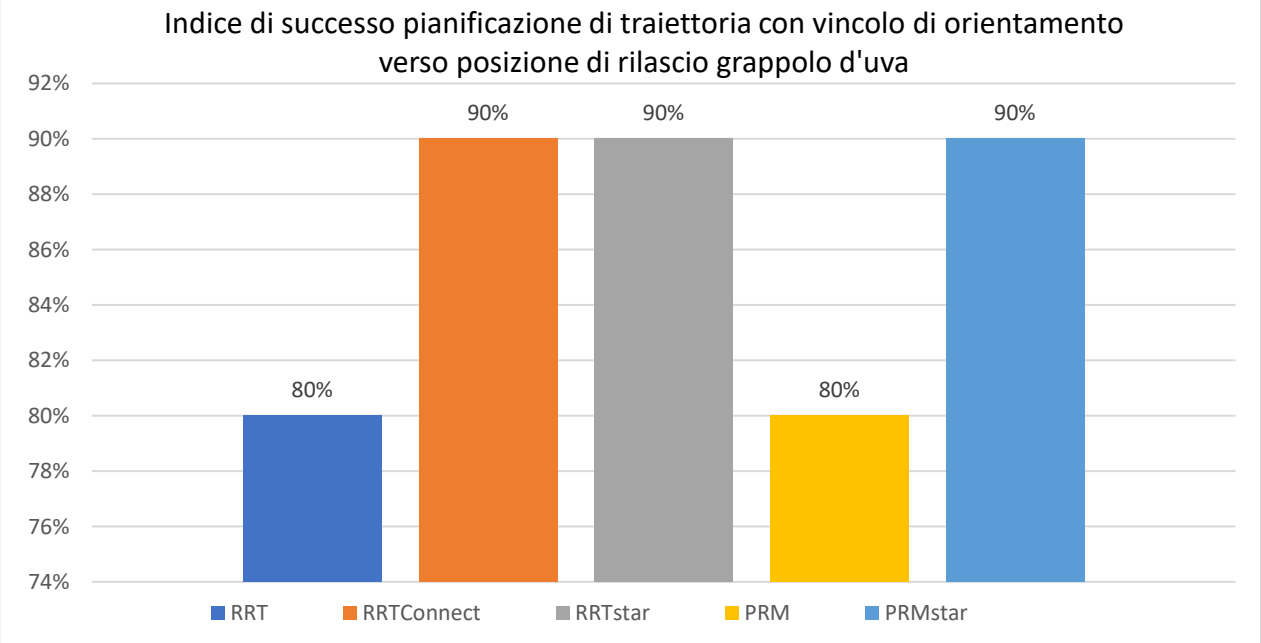


Figura 50 - Sintesi dei tentativi di pianificazione andati a buon fine per la Prova – 3 dei diversi pianificatori.

Prova – 4

γ (angolo di beccheggio) =0	X [m]	Y [m]	Z [m]
Distanza tra i due filari [m]		2	
Raccolta grappolo SR Root (base del robot)	0,45	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15

Tabella 14 - Dati raccolta e rilascio grappolo Prova – 4.

I risultati dei pianificatori *RRT* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
RRT	Raccolta	10	100%	0,0367286
	Rilascio	10	100%	10,7735328
RRT Connect		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,025043
	Rilascio	10	70%	10,941737
RRT star		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,1003835
	Rilascio	10	90%	25,01388756

Tabella 15 - Risultati Prova – 4 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori RRT, RRTConnect e RRTstar.

I risultati dei pianificatori *PRM* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
PRM	Raccolta	10	100%	0,1053298
	Rilascio	10	50%	15,0281164
PRMstar		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,1074717
	Rilascio	10	90%	15,03206478

Tabella 16 - Risultati Prova – 4 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori PRM e PRMstar.

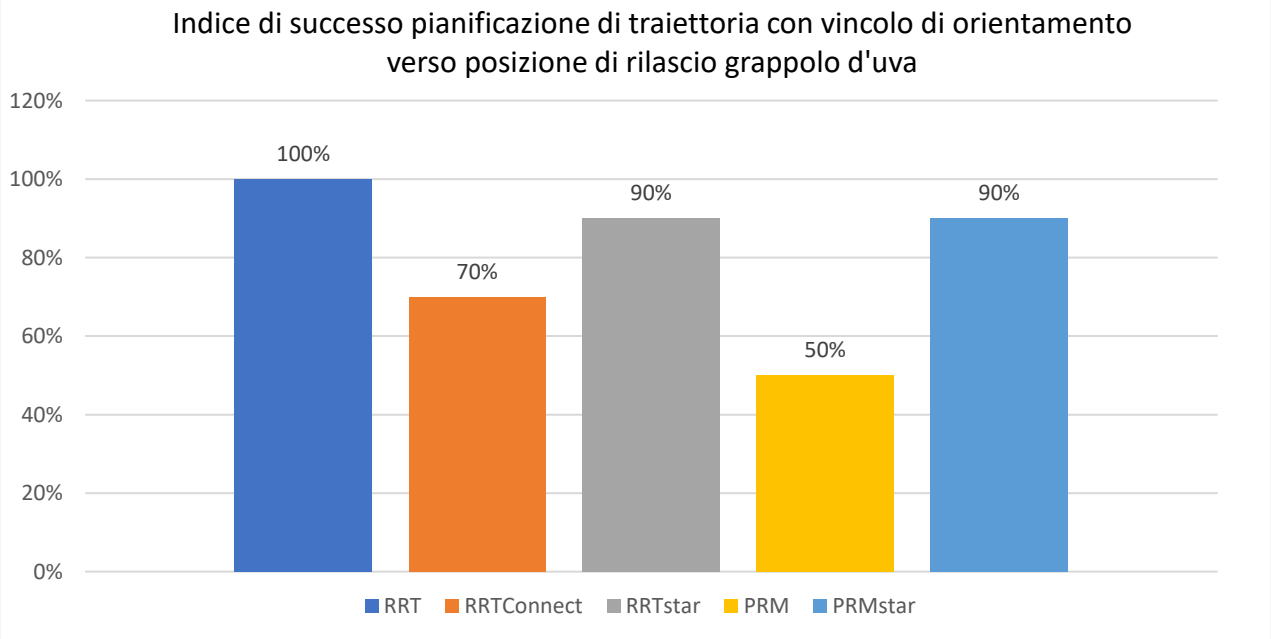


Figura 51 - Sintesi dei tentativi di pianificazione andati a buon fine per la Prova – 4 dei diversi pianificatori.

Prova – 5

γ (angolo di beccheggio) =0	X [m]	Y [m]	Z [m]
Distanza tra i due filari [m]		2	
Raccolta grappolo SR Root (base del robot)	-0,45	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15

Tabella 17 - Dati raccolta e rilascio grappolo Prova – 5.

I risultati dei pianificatori *RRT* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
RRT	Raccolta	10	100%	0,0320596
	Rilascio	10	80%	15,37577363
RRT Connect		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,0274285
	Rilascio	10	100%	13,878371
RRT star		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,1009056
	Rilascio	10	80%	25,01273913

Tabella 18 - Risultati Prova – 5 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori RRT, RRTConnect e RRTstar.

I risultati dei pianificatori *PRM* raccolti a valle delle prove sono riportati nella tabella seguente:

		#Prove	Indice di successo	t pianificaz [s]
PRM	Raccolta	10	100%	0,1080378
	Rilascio	10	60%	15,03028267
PRMstar		#Prove	Indice di successo	t pianificaz [s]
	Raccolta	10	100%	0,10626131
	Rilascio	10	90%	15,02652556

Tabella 19 - Risultati Prova – 5 pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatori PRM e PRMstar.

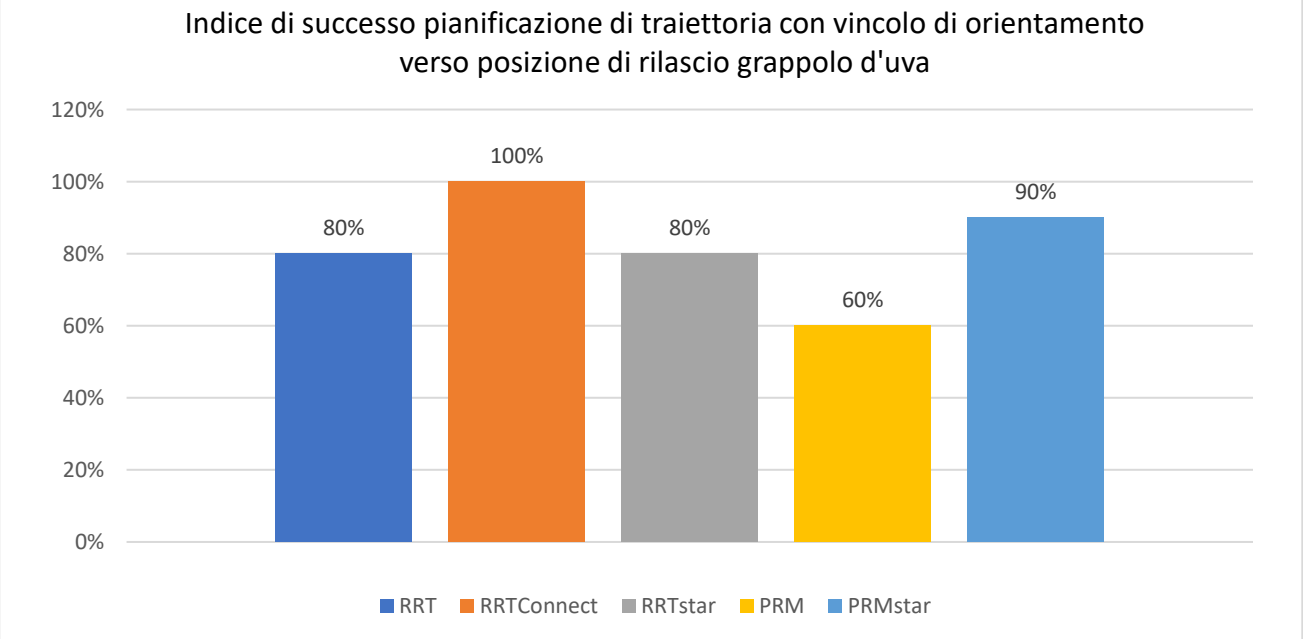


Figura 52- Sintesi dei tentativi di pianificazione andati a buon fine per la Prova – 5 dei diversi pianificatori.

A valle delle prove condotte è possibile appurare che il pianificatore che meglio si presta all'applicazione di raccolta e rilascio grappolo d'uva con vincoli di orientamento è il pianificatore ***PRMstar***.

PRMstar è il pianificatore che presenta il minor tempo medio per la pianificazione di traiettorie con vincoli di orientamento, con un indice di successo globale pari all'86% con 43 pianificazioni andate a buon fine a fronte di 50 tentativi. Il tempo medio per la pianificazione è stato di 15.04 sec.

A seguire ci sono:

- ***RRTConnect***: con un indice di successo globale pari a 84% con 42 pianificazioni andate a buon fine a fronte di 50 tentativi. Il tempo medio per la pianificazione è stato di 13.84 sec;
- ***RRT***: con un indice di successo globale pari a 78% con 39 pianificazioni andate a buon fine a fronte di 50 tentativi. Il tempo medio per la pianificazione è stato di 14.77 sec;
- ***PRM***: con un indice di successo globale pari a 74% con 37 pianificazioni andate a buon fine a fronte di 50 tentativi. Il tempo medio per la pianificazione è stato di 15.03 sec;
- ***RRTstar***: con un indice di successo globale pari a 72% con 36 pianificazioni andate a buon fine a fronte di 50 tentativi. Il tempo medio per la pianificazione è stato di 25.02 sec;

I risultati ottenuti sono più o meno in linea con quelle che erano le aspettative teoriche.

Nonostante i pianificatori *RRT* vengano maggiormente impiegati per la loro rapidità nel trovare una soluzione valida a problemi di pianificazione del moto in presenza di ostacoli, il pianificatore che ha presentato delle prestazioni migliori per la nostra applicazione è stato *PRMstar*. Questo potrebbe essere dovuto al fatto che lo spazio di lavoro C è altamente vincolato e pertanto, la densità di nodi nella *RoadMap* in prossimità di "spazi stretti" o in prossimità dei bordi è maggiore.

I pianificatori del percorso *RRT* ed *RRTConnect* durante i 25 secondi che gli vengono forniti per la pianificazione, terminano la loro ricerca nel momento in cui trovano una soluzione. Per questo motivo, il tempo medio necessario alla ricerca del percorso, è molto minore rispetto al pianificatore *RRTstar* che, invece, sfrutta tutto il tempo che gli viene fornito. L'algoritmo *RRTstar,* infatti, prevede che iterativamente esso continui a cercare il percorso più breve per andare dalla configurazione di raccolta grappolo a quella di rilascio grappolo.

Per quanto riguarda i pianificatori di percorsi *PRM*, il risultato era in linea con le aspettative.

Entrambi questi pianificatori, *PRM* e *PRMstar*, prevedono la generazione di una *RoadMap* e pertanto entrambi sfruttano tutto il tempo che gli viene concesso per cercare e salvare delle configurazioni valide all'interno dello spazio di lavoro C_{free}. *PRMstar* è un algoritmo tale per cui la densità delle configurazioni negli spazi stretti (in prossimità di ostacoli e/o limiti di giunto), è più alta e quindi è chiaro che la possibilità di cercare un percorso valido in un ambiente vincolato e ricco di ostacoli è maggiore.

Area di pianificazione ottimale per PRMstar

Una volta individuato ***PRMstar*** come il pianificatore di percorsi ottimale per l'applicazione oggetto di studio, si sono valutate le sue prestazioni all'interno dell'area di lavoro del manipolatore.

Mantenendo costante:

- Il tempo concesso per la pianificazione delle traiettorie con vincolo di orientamento, pari a 15 secondi;
- Risolutore di cinematica inversa → `Track-IK solve_type: Manipulation2;`
- Parametrizzazione nel tempo → `Iterative Parabolic Time Parameterization (IPTP)`
- la quota Y = 0.5 m di raccolta grappolo;
- la "Posizione di rilascio grappolo";

Si è variata la "Posizione di raccolta grappolo" all'interno dell'area di lavoro del manipolatore in modo da valutare l'area di lavoro ottimale per l'esecuzione delle operazioni oggetto di studio.

Descrizione	X [m]	Y [m]	Z [m]
Raccolta grappolo SR Root (base del robot)	-0,68 < X < 0,68	0,5	0,275 < Z < 0,95

Tabella 20 - Dati di raccolta grappolo per l'individuazione dell'area ottimale di lavoro a Y=0.5 m rispetto alla base nel manipolatore.

Nell'immagine riportata di seguito è possibile apprezzare:

- Area di lavoro del manipolatore approssimata ad Y = 0.5 m;
- Area di lavoro del manipolatore con vincolo di orientamento ad Y = 0.5 m;

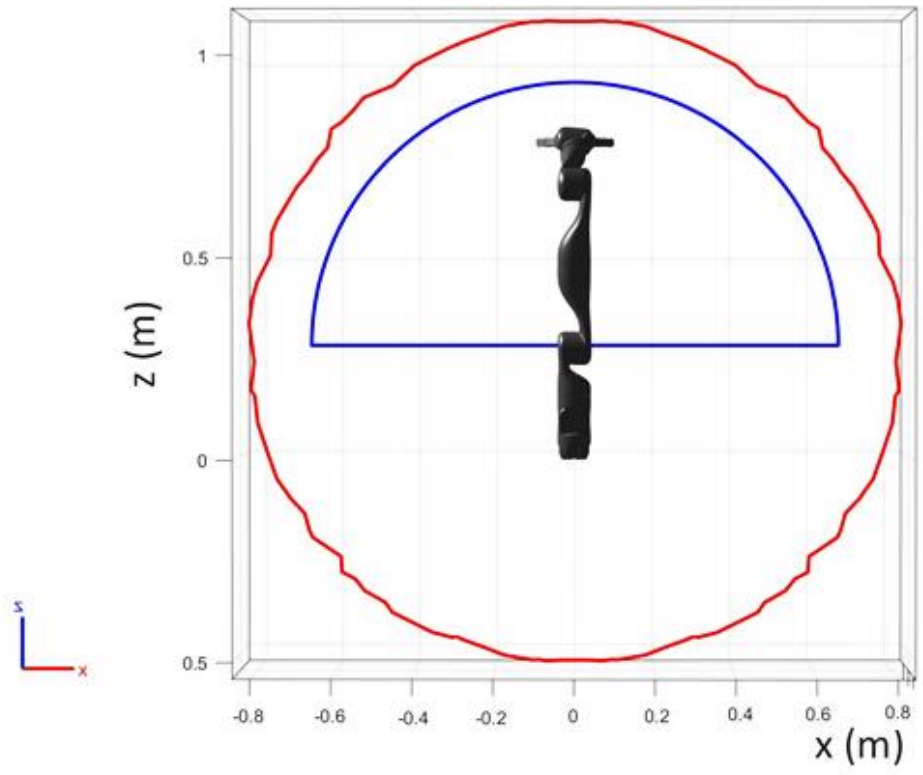


Figura 53 - Illustrazione approssimata delle aree di lavoro del manipolatore ad una quota di Y = 0.5 m rispetto al SR situato alla base del manipolatore

Di seguito i risultati ottenuti durante le prove:

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	-0,68	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,10807445
Rilascio	10	80%	15,01344788

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	-0,45	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,106255
Rilascio	10	80%	15,01627825

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	-0,3	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1074628
Rilascio	10	70%	15,02963586

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1100654
Rilascio	10	90%	15,04778956

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0,3	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1063769
Rilascio	10	70%	15,01685971

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0,45	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1075204
Rilascio	10	80%	15,03830925

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0,68	0,5	0,275
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1071385
Rilascio	10	70%	15,05270963

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	-0,6	0,5	0,5
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1074828
Rilascio	10	60%	15,03994433

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	-0,3	0,5	0,5
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1073722
Rilascio	10	80%	15,01742663

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0	0,5	0,5
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1069535
Rilascio	10	90%	15,019471

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0,3	0,5	0,5
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1065143
Rilascio	10	100%	15,0211014

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0,6	0,5	0,5
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,107202
Rilascio	10	100%	15,0312696

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	-0.45	0,5	0,7
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,10626131
Rilascio	10	90%	15,02652556

	X [m]	Y [m]	Z [m]
Raccolta grappolo SR Root (base del robot)	-0.30	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,106565
Rilascio	10	90%	15,04077

	X [m]	Y [m]	Z [m]
Raccolta grappolo SR Root (base del robot)	0	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1969619
Rilascio	10	60%	15,02532822

	X [m]	Y [m]	Z [m]
Raccolta grappolo SR Root (base del robot)	0.30	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,096636
Rilascio	10	100%	15,0515311

	X [m]	Y [m]	Z [m]
Raccolta grappolo SR Root (base del robot)	0.45	0,5	0,7
Rilascio uva in un cesto SR Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1074717
Rilascio	10	90%	15,03206478

	X [m]	Y [m]	Z [m]
Raccolta grappolo RF Root (base del robot)	0	0,5	0,95
Rilascio uva in un cesto RF Root (base del robot)	0	-0,5	0,15
	#Prove	**Indice di successo**	**t pianificaz [s]**
Raccolta	10	100%	0,1061978
Rilascio	10	90%	15,02459622

Tabella 21 – Risultati di pianificazione di traiettorie di raccolta e rilascio grappolo d'uva. Pianificatore di percorsi PRMstar.

A valle delle prove condotte è stato possibile individuare un'area di lavoro ottimale per la pianificazione dei percorsi con vincolo di orientamento, per il pianificatore *PRMstar*, ad una distanza pari a Y= 0.5 m rispetto al SR base del manipolatore.

Aver individuato quest'area di lavoro potrebbe risultare utile su campo.

Muovendosi lungo il filare di un vigneto, oppure, azionando il dispositivo di posizionamento di *Agri.q02*, sarà possibile far rientrare il grappolo d'uva (o qualsiasi altro oggetto dalle medesime dimensioni) all'interno dell'area di lavoro ottimale per la pianificazione di percorsi con vincolo di orientamento.

Nell'immagine di seguito riportata si possono notare tre diverse aree di interesse per la specifica applicazione:

- Area di lavoro del manipolatore approssimata a una quota Y= 0.5 m;
- Area di lavoro del manipolatore con vincolo di orientamento a una quota Y= 0.5 m;
- Area di pianificazione ottimale per il pianificatore *PRMstar* a una quota Y= 0.5 m (indice di successo ≥ 90%);

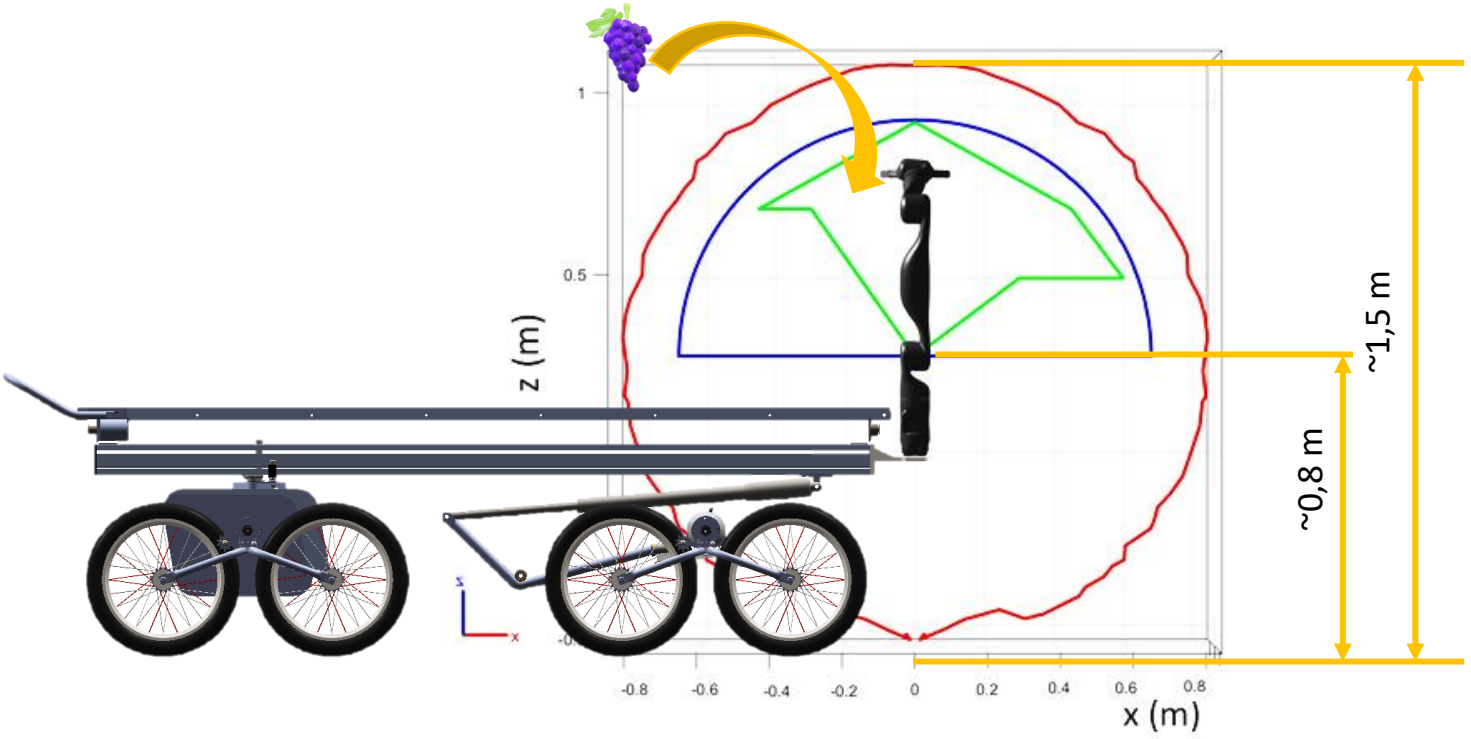


Figura 54 - Illustrazione del manipolatore Kinova Jaco2 installato sul rover Agri.q02. Sono riportate in colori differenti le aree di lavoro del manipolatore ad una distanza Y = 0.5 m rispetto al SR base.

5. Conclusioni

Il presente lavoro di tesi nasce con l'obiettivo di valutare se, attraverso l'utilizzo di *ROS* e quindi di *MoveIt*, sia possibile controllare il manipolatore *Kinova Jaco2* in maniera adeguata, per l'esecuzione di operazioni di raccolta di campioni all'interno di vigneti.

Pertanto, durante il lavoro svolto, è stato necessario approfondire la conoscenza e le competenze relative al framework *ROS* e quelle relative alla programmazione ad oggetti tipica del linguaggio programmazione *C++.* È stato inoltre indispensabile studiare la struttura delle librerie fornite dagli sviluppatori *Kinova*.

Attraverso un'analisi comparativa dei pianificatori di percorsi implementati dalle librerie OMPL è stato possibile affrontare alcuni argomenti basilari relativi alla pianificazione di traiettorie in presenza di ostacoli.

Le librerie *OMPL* implementano pianificatori del moto basati sul campionamento e permettono di generare percorsi validi, privi di collisioni, in presenza di ostacoli. Risulta però necessario definire in maniera molto accurata l'ambiente entro il quale il robot o il manipolatore deve muoversi, al fine di evitare, un'eventuale collisione con gli oggetti che lo circondano.

A valle delle analisi condotte, si è individuato il miglior pianificatore di traiettorie con vincoli di orientamento, che meglio si adatta alla specifica applicazione e che potrebbe essere impiegato sul campo.

La scelta è ricaduta su ***PRMstar,*** poiché questo pianificatore presenta le migliori prestazioni in termini di tempo medio di pianificazione ed elevate percentuali di successo (83% di successi su 180 pianificazioni totali). Essendo lo spazio di lavoro *C*, altamente vincolato, la densità di nodi nella *RoadMap* in prossimità di "spazi stretti" o in prossimità dei bordi è maggiore e pertanto è più probabile che venga trovata una soluzione alla richiesta di pianificazione del moto.

Inoltre, ad una distanza pari a Y = 0.5 m, è stata individuata un'area di lavoro ottimale per la pianificazione di percorsi con vincoli di orientamento. Aver individuato quest'area di lavoro potrebbe risultare utile su campo: infatti, muovendosi lungo il filare di un vigneto, oppure, azionando il dispositivo di posizionamento di *Agri.q02*, sarà possibile far rientrare il grappolo d'uva, o qualsiasi altro oggetto delle medesime dimensioni, all'interno dell'area di lavoro ottimale.

Il controllo del manipolatore reale con *MoveIt* presenta dei limiti non trascurabili, poiché, essendo un framework impiegato per lo sviluppo di generiche applicazioni robotiche non risulta molto efficace per l'applicazione oggetto di studio.

Nello specifico, non vi è un ottimale adeguamento dell'*API* fornita dagli sviluppatori *Kinova* per la gestione di pianificazione di traiettorie con vincoli di orientamento. Infatti, se non viene espressamente indicata la pianificazione dei percorsi in spazio giunti, si perde completamente il controllo del manipolatore rischiando la collisione di quest'ultimo con gli oggetti che ha attorno.

Non di minore importanza è la necessità di un tempo computazionale elevato per la ricerca di un percorso valido e privo di collisioni per l'esecuzione di una traiettoria con vincoli di orientamento. Sebbene il nodo `move_group` sia in grado di trovare una traiettoria valida in un ambiente fortemente vincolato, il tempo medio richiesto per la pianificazione è di circa 15 secondi ai quali va sommato il tempo per l'esecuzione della traiettoria.

Inoltre, il nodo `move_group` non garantisce una perfetta esecuzione delle traiettorie pianificate. In fase di pianificazione delle traiettorie, l'adattatore *"AddTimeParameterization"* raccorda con un andamento parabolico due tratti lineari con diverso coefficiente angolare. Pertanto, durante l'esecuzione della traiettoria, il manipolatore, non esegue perfettamente la traiettoria pianificata. Il rischio è che il manipolatore reale vada a collidere con gli oggetti cha ha attorno, anche se, la traiettoria pianificata risulta essere priva di collisioni.

Per i motivi appena esposti, allo stato attuale, le librerie *OMPL* non sono direttamente implementabili per il controllo del manipolatore installato sul prototipo di *Agri.q02*.

Tra gli sviluppi futuri di questa applicazione sicuramente sono auspicabili:

- l'implementazione di nuovi algoritmi di parametrizzazione nel tempo;
- la gestione dell'angolo di beccheggio del pannello solare attraverso il framework *ROS*;
- l'installazione di un sistema di visione in prossimità dell'organo terminale del manipolatore (in configurazione *eye-on-hand*), impiegato per il riconoscimento di un grappolo d'uva o di un qualsiasi oggetto delle medesime dimensioni. Sarebbe così possibile integrare il sistema di visione all'interno dell'ambiente di simulazione ***rviz*** per verificare eventuali collisioni del manipolatore con ciò che lo circonda;
- la realizzazione di un organo terminale appositamente realizzato per la raccolta di un grappolo d'uva in modo che possa contemporaneamente tagliare il picciolo del grappolo, come una forbice da potatura, e raccogliere il grappolo come la mano di un uomo.

Bibliografia e Sitografia

[1]- James Underwood, A. W. (2017). Efficient in-field plant phenomics for row-crops with an autonomous ground vehicle. (A. W. - James Underwood, A cura di) *Journal of Field Robotics*, 23. doi:10.1002/rob.21728;

[2]- Boaz Arad, J. B.-S. (2020). Development of a sweet pepper harvesting robot. (J. B.-S. - Boaz Arad, A cura di) *Journal of Field Robotics*, 13. doi:10.1002/rob.21937;

[3]- Ferran Roure, G. M. (2017). GRAPE: Ground Robot for vineyArd. *ROBOT 2017: Third Iberian Robotics Conference*, 249-260. doi:10.1007/978-3-319-70833-1_21;

[4] - Giuseppe Quaglia, C. V. (2019). Design of the positioning mechanism of an unmanned ground vehicle for precision agriculture. *Advances in Mechanism and Machine Science* , 3531-3540. doi:10.1007/978-3-030-20131-9_348;

[5] - Manuale d'uso e manutenzione Kinova Gen2 7 DoF. (s.d.). Gen2 7 DoF User Guide. Tratto da https://www.kinovarobotics.com/en/resources/gen2-technical-resources;

[6] - Y. Pyo, H. C. (2017). *ROS Robot Programming. ROBOTIS.* ;

[7] - International Federation of Robotics, IFR. (s.d.). *Robot di servizio*. Tratto da International Federation of Robotics: https://ifr.org/service-robots;

[8] - MoveIt! (s.d.). *MoveIt!* Tratto da https://moveit.ros.org/documentation/concepts/

[9] - WikiROS, BSD. (s.d.). *ROS.org*. Tratto da http://wiki.ros.org/it;

[10] - Robotica. modellistica, pianificazione e controllo. (2008). Robotica. modellistica, pianificazione e controllo. In S. V. - Siciliano, *Robotica. modellistica, pianificazione e controllo* (p. 640). McGraw-Hill Education.;

[11] - Kevin M. Lynch and Frank C.Park,. (2017). *MODERN ROBOTICS - Mechanics, Planning and Control.* Cambridge University Press.;

[12] - Kavraki Lab Rice University,. (2021). Open Motion Planning Library: A Primer. 25. Tratto da http://ompl.kavrakilab.org/;

[13] - Fang Yuan, Liang, Jia-Hong; Fu, Yue-Wen; Xu, Han-Cheng; Ma, Ke. (2015). A hybrid sampling strategy with optimized Probabilistic Roadmap Method. *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2298-2302. doi: 10.1109/FSKD.2015.7382311;

[14] - Patrick Beeson and Barrett Ames, . (2015). TRAC-IK: An Open-Source Library for. *15th International Conference on Humanoid Robots*.;

[15] - MoveIt. (s.d.). *MoveIt - Melodic*. Tratto da http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/ompl_interface/ompl_interface_tutorial.html;

[16] - Tsuneo Yoshikawa. (1985). Manipulability of Robotic Mechanisms. *SAGE*. doi:10.1177/027836498500400201;

[17] - TRACLABS. (s.d.). *Public repository for the current stable version of the TRACLabs' IK solver*. Tratto da bitbucket.org: https://bitbucket.org/traclabs/trac_ik/src/master/;

[18] - J. Kenneth Salisbury, John J. Craig. (1982). Articulated Hands: Force Control and Kinematic Issues. *The International Journal of Robotics Research*. doi:10.1177/027836498200100102;

[19] - Stephen L. Chiu. (1988). Task Compatibility of Manipulator Postures. *The International Journal of Robotics Research - SAGE*. doi:10.1177/027836498800700502;

[20] - XML - Link. (s.d.). Tratto da WikiROS: http://wiki.ros.org/urdf/XML/link;

[21] - Tobias Kunz and Mike Stilman - Georgia Institute of Technology. (s.d.). Time-Optimal Trajectory Generation for Path. (T. K. Stilman);

[22] - Tobias Kunz and Mike Stilman - Turning Paths Into Trajectories Using Parabolic Blends. (2011). Turning Paths Into Trajectories Using Parabolic Blends.;

[23] - XML - Joint. (s.d.). *WikiROS*. Tratto da WikiROS: http://wiki.ros.org/urdf/XML/joint;
[24] - MoveIt ROS Planning Interface. (s.d.). Tratto da http://docs.ros.org/en/jade/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1MoveGroup.html;
[25] - MoveIt Visual Tools. (s.d.). Tratto da http://docs.ros.org/en/kinetic/api/moveit_visual_tools/html/classmoveit__visual__tools_1_1MoveItVisualTools.html;
[26] - ROS Control. (s.d.). Tratto da http://wiki.ros.org/ros_control;
[27] - Jia Pan, Sachin Chitta, Dinesh Manocha -. (2012). FCL: A General Purpose Library for Collision and Proximity Queries. doi:10.1109/ICRA.2012.6225337

Appendice - A

Di seguito vi è il codice in linguaggio MATLAB che riporta le trasformazioni di coordinate dal SR Base del robot fino al SR nell'organo terminale:

```
clc
clear all

%q=[0 0 0 0 0 0 0];  %%Cfg 0-Assi impossibile da raggiungere

q=[4.9409 2.8396 0.0016 0.7581 4.6342 4.4963 5.0252]; %Configurazione di Home
nello spazio giunti
%q=[0 pi 0 pi 0 pi 0]; %Robot completamente esteso
%q=[2 1.5 2.5 pi 2.8 1.2 2.3]; %Configurazione random

%Caratteristiche geometriche
d1=275.5;
d3=410;
d4=13.3;
d5=311.1;
d7=263.8;

%Matrici di trasformazione dei SR rispetto al link precedente partendo dal SR-
Base fino al SR centrato nell'EE.

A_B_0=[1    0   0    0;         % Trasformazione del SR da BASE a O
       0   -1   0    0;
       0    0  -1   0;
       0    0    0   1];

A_0_1=[cos(q(1,1)+ pi)    0   sin(q(1,1)+ pi)    0; % Trasformazione del SR da
       sin(q(1,1)+ pi)    0   -cos(q(1,1)+ pi)   0; % da 0 a 1
       0                  1         0           -d1;
       0                  0         0            1];



A_1_2=[cos(q(1,2))       0   sin(q(1,2))    0;    % Trasformazione del SR da 1 a 2
       sin(q(1,2))       0   -cos(q(1,2))   0;
       0                 1         0        0;
       0                 0         0        1];

A_2_3=[cos(q(1,3))       0   sin(q(1,3))    0;    % Trasformazione del SR da 2 a 3
       sin(q(1,3))       0   -cos(q(1,3))   0;
       0                 1         0     -d3;
       0                 0         0       1];


A_3_4=[cos(q(1,4))       0   sin(q(1,4))    0;    % Trasformazione del SR da 3 a 4
       sin(q(1,4))       0   -cos(q(1,4))   0;
       0                 1         0     -d4;
       0                 0         0       1];


A_4_5=[cos(q(1,5))       0   sin(q(1,5))   0;     % Trasformazione del SR da 4 a 5
       sin(q(1,5))       0   -cos(q(1,5))  0;
       0                 1         0     -d5;
       0                 0         0       1];


A_5_6=[cos(q(1,6))       0   sin(q(1,6))   0;     % Trasformazione del SR da 5 a 6
```

```
       sin(q(1,6))       0    -cos(q(1,6)) 0;
       0                 1          0      0;
       0                 0          0      1];

A_6_7=[cos(q(1,7)+ pi/2)  sin(q(1,7)+ pi/2)     0    0;      % Trasformazione del
       sin(q(1,7)+ pi/2)  -cos(q(1,1)+ pi/2)    0    0;      % SR da 6 a 7
       0                    0                 -1    -d7;
       0                    0                  0     1];



% A_B_1=A_00_B*A_B_0*A_0_1;
% A_0_2=A_00_B*A_B_0*A_0_1*A_1_2;
% A_0_3=A_00_B*A_B_0*A_0_1*A_1_2*A_2_3;
% A_0_4=A_00_B*A_B_0*A_0_1*A_1_2*A_2_3*A_3_4;
% A_0_5=A_00_B*A_B_0*A_0_1*A_1_2*A_2_3*A_3_4*A_4_5;
% A_0_6=A_00_B*A_B_0*A_0_1*A_1_2*A_2_3*A_3_4*A_4_5*A_5_6;
% A_0_7=A_00_B*A_B_0*A_0_1*A_1_2*A_2_3*A_3_4*A_4_5*A_5_6*A_6_7;

A_B_1=A_B_0*A_0_1;
A_0_2=A_B_0*A_0_1*A_1_2;
A_0_3=A_B_0*A_0_1*A_1_2*A_2_3;
A_0_4=A_B_0*A_0_1*A_1_2*A_2_3*A_3_4;
A_0_5=A_B_0*A_0_1*A_1_2*A_2_3*A_3_4*A_4_5;
A_0_6=A_B_0*A_0_1*A_1_2*A_2_3*A_3_4*A_4_5*A_5_6;
A_0_7=A_B_0*A_0_1*A_1_2*A_2_3*A_3_4*A_4_5*A_5_6*A_6_7;


p_0_1=[0  0  -d1 1]';
p_1_2=[0  0   0  1]';
p_2_3=[0  0  -d3 1]';
p_3_4=[0  0  -d4 1]';
p_4_5=[0  0  -d5 1]';
p_5_6=[0  0   0  1]';
p_6_7=[0  0  -d7 1]';

p1=A_0_1*[0 0 0 1]'; %OK
p2=A_0_2*[0 0 0 1]'; %OK
p3=A_0_3*[0 0 0 1]'; %OK
p4=A_0_4*[0 0 0 1]'; %OK
p5=A_0_5*[0 0 0 1]'; %OK
p6=A_0_6*[0 0 0 1]'; %OK
p7=A_0_7*[0 0 0 1]' %OK

p_0_3=A_0_2*p_2_3;

%se il braccio è tutto esteso z_0_7=1260.4 mm; z_0_5=996.6 mm; z_0_4=892.8mm

%Posizione del Tool nello spazio operativo
p_0_7=A_0_6*p_6_7

%Posizione del centro del polso nello spazio operativo
p_0_5=A_0_4*p_4_5;
```

Appendice – B

```
planner_configs:
  RRTkConfigDefault:
    type: geometric::RRT
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
    enforce_joint_model_state_space: true
  RRTConnectkConfigDefault:
    type: geometric::RRTConnect
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    enforce_joint_model_state_space: true
  RRTstarkConfigDefault:
    type: geometric::RRTstar
    range: 0.0  # Max motion added to tree. ==> maxDistance_ default: 0.0, if 0.0, set on setup()
    goal_bias: 0.05  # When close to goal select goal, with this probability? default: 0.05
    delay_collision_checking: 1  # Stop collision checking as soon as C-free parent found. default 1
    enforce_joint_model_state_space: true
  PRMkConfigDefault:
    type: geometric::PRM
    max_nearest_neighbors: 10  # use k nearest neighbors. default: 10
    enforce_joint_model_state_space: true
  PRMstarkConfigDefault:
    type: geometric::PRMstar
    enforce_joint_model_state_space: true
```