UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITA' DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Sviluppo e implementazione di una interfaccia grafica in realtà aumentata per il controllo semplificato di sistemi robotici da remoto o in ambiente ostile

Relatore: Prof. Stefano Debei

Laureando: Marco Reschiglian

matr. 1129292

Anno Accademico 2017/2018

SOMMARIO

Il questo lavoro di tesi, verrà esposto il processo di realizzazione dell'interfaccia grafica di comando del rover studentesco MORPHEUS. Verranno motivate le necessità evidenziate per lo sviluppo di tale lavoro, le scelte di design e si specificheranno i requisiti e i vincoli realizzativi. In una seconda parte, verrà analizzata la costruzione di un modello in ambiente virtuale del braccio robotico del rover, e di come sarà possibile integrarlo per permettere la pianificazione di traiettoria da parte dell'operatore tenendo conto dello scenario operativo in tempo reale. Infine verranno commentate possibili integrazioni ed evoluzioni del lavoro svolto.

INDICE ANALITICO

INDICE DELLE FIGURE

CAP. I - IL ROVER STUDENTESCO MORPHEUS

Il progetto MORPHEUS (Mars Operative Rover of Padova Engineering Students) è un'iniziativa studentesca che, sotto l'egida del Prof. Stefano Debei, direttore del Centro di Ateneo di Studi e Attività Spaziali "G.Colombo" (CISAS), si pone come obiettivo la progettazione e la realizzazione di un rover dedicato a mansioni affini a quelle di esplorazione planetaria. In pratica, il rover punta a diventare una piattaforma di sperimentazione didattica per il corso di Robotica Spaziale tenuto dal Prof. Debei per gli studenti magistrali di Ingegneria Aerospaziale, e come banco di prova per studenti di ingegneria di molte specialità diverse, provenienti dall'area Industriale e dell'Informazione, in parallelo con lo studio teorico delle materie d'esame. Con la collaborazione del Prof. Marco Pertile, dell'Ing. Sebastiano Chiodini, dell'Ing. Riccardo Giubilato e di altri dottorandi, dalla fine del 2014 un team di studenti (rinnovato ogni anno) si occupa della progettazione del rover, della sua realizzazione e del continuo miglioramento. L'obiettivo risulta nella realizzazione completa di un veicolo a controllo remoto (e con funzionalità di guida autonoma) che abbia capacità di movimento, localizzazione, mappatura e guida autonoma su terreni accidentati, raccolta di campioni di interesse geologico e scientifico, ed eventualmente come supporto per operazioni di assistenza e manutenzione in un'ipotetica operazione di esplorazione planetaria.

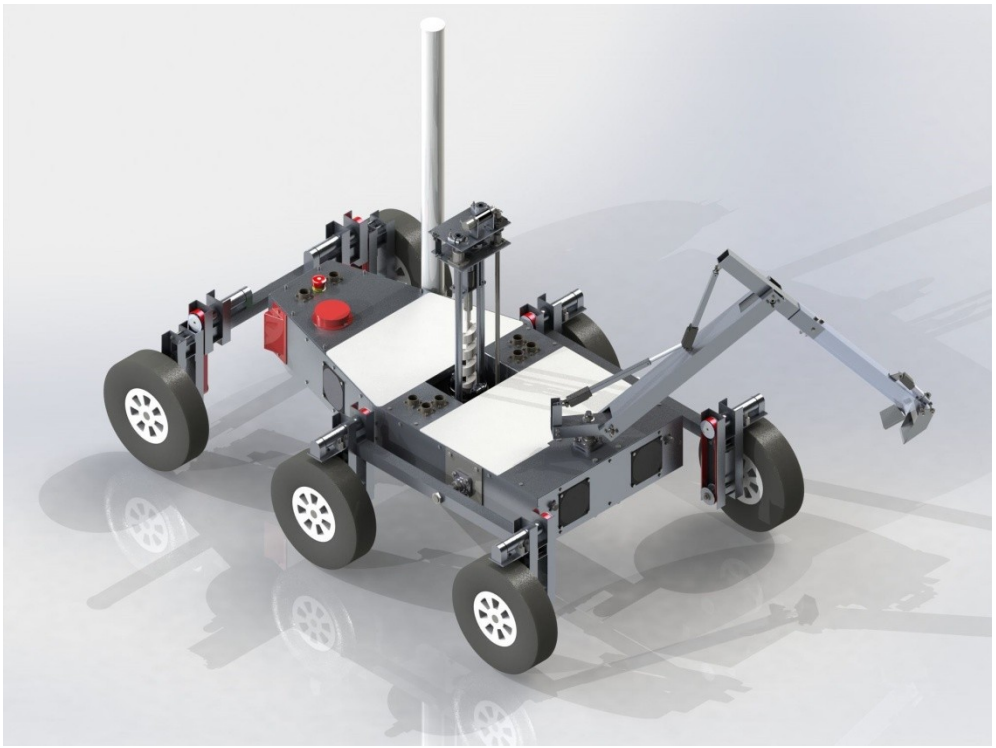


Figura 1.1. *Rappresentazione del rover MORPHEUS nella prima iterazione di design*

Un altro ambizioso obiettivo del gruppo è infine quello di partecipare ad una competizione internazionale di robotica studentesca, in particolare la competizione ERC (European Rover Challenge) che si svolge in Polonia e sfida gruppi universitari alla costruzione e all'impiego di rover in prove che ricalcano le mansioni sopra descritte.
Nel corso di questi anni il gruppo ha riunito studenti di ingegneria provenienti da 5 corsi di laurea differenti, partendo dalla progettazione del primo design e realizzazione dei pezzi nel 2015. L'assemblaggio di meccanica ed elettronica nel 2016 rileva alcuni problemi che rendono necessarie profonde correzioni e i test procedono fino a che, a fine 2017, i principali sottosistemi del rover diventano operativi e il rover MORPHEUS diventa una delle attrazioni dello stand del CISAS nella Notte Europea della Ricerca organizzata dall'Università di Padova a fine Settembre 2017.

Il lavoro di questa tesi, che si innesta nell'ambito del software necessario al funzionamento del rover, nasce dalla necessità di unire in maniera organica le componenti software sviluppate nel corso degli anni per i vari sottosistemi e organizzarle in un unico strumento dedicato per l'operatore che, nel caso di operazioni non autonome, andrà a guidare il rover a distanza (senza poterlo vedere) da un computer portatile connesso al rover tramite antenna, mutuando le attuali operazioni di esplorazione planetaria sulla superficie di Marte. L'avvio della sequenza di operazioni necessarie nei vari sottosistemi, le azioni per impartire i comandi che sono note ai programmatori del singolo sottosistema, risultano inefficaci nell'impiego del rover tramite stringhe di comando da terminale; i dati dei sensori e le informazioni di ritorno, codificate in forma numerica per poter essere eseguite dai programmi di controllo, non sono funzionali alla guida di un veicolo complesso. Tutti questi fattori hanno dato quindi impulso all'inizio di un lavoro di riorganizzazione dell'impianto software che si è concretizzato nella necessità di realizzare un'interfaccia grafica per il comando semplificato del rover.

Contestualmente nel 2017 è iniziata una revisione più significativa della parte fisica del rover, sfociata nell'iniziativa denominata "Padova University Students Geological Samples Robotic Arm Collector", finalizzata ad una completa riprogettazione del braccio che ha influenzato lo svolgimento della seconda parte di questo lavoro di tesi.

CAP. II - SOFTWARE e INTERFACCIA

2.1 Software di MORPHEUS

Il software di Morpheus si basa sulla struttura open-source di ROS (Robot Operating System). ROS è un framework sviluppato a partire dal 2007 che si pone come obiettivo di creare un'infrastruttura modulare per lo sviluppo di software di controllo per robot. Per poter meglio comprendere il contesto in cui si inserisce il lavoro di tesi, ROS verrà descritto in breve all'inizio di questo capitolo, per poter poi descrivere nel dettaglio l'apparato software che governa Morpheus.

2.2 Struttura di ROS

Di norma il controllo software di un robot richiede una serie di entità software distinte, ciascuna dedicata alla generazione di opportuni segnali per l'azionamento dei singoli attuatori. Ognuna di queste entità può, in linea teorica, essere eseguita indipendentemente dalle altre in quanto non comunicanti; nel funzionamento reale di un sistema tuttavia ciascuna di queste entità deve essere eseguita contemporaneamente alle altre, o meglio eseguita ciclicamente in tempi brevissimi se l'esecuzione di processi in parallelo non è praticabile. All'interno di ROS è possibile scrivere ognuna di queste entità come singoli programmi, e all'interno della struttura di ROS questi programmi sono eseguiti in maniera sincronizzata con un processo di controllo globale, definito ***master*** all'interno di ROS.

Ciascuna dei singoli processi rappresenta un **nodo** di ROS, sincronizzato rispetto al clock definito dal nodo ***master***. Ciascuno di questi nodi potrebbe essere eseguito su macchine diverse, ad esempio una serie di computer collegati via LAN o comunicanti attraverso un'antenna wireless. Il master di ROS provvede ad instaurare un protocollo di tipo P2P tale da poter eseguire ciascun nodo solo nell'hardware predisposto.

Nel caso di robot reali, è probabile che alcuni di questi nodi debbano comunicare tramite l'invio di opportuni segnali, in ROS definiti messaggi; a causa della distribuzione dei nodi risulta necessario definire un sistema che permetta tale comunicazione. ROS definisce in questo caso delle stringhe contenenti il messaggio (e altre meta-informazioni) dette **topic**: ciascun nodo può inviare il suo messaggio pubblicandolo sul detto topic, o riceverlo leggendo

il susseguirsi delle variazioni del topic interessato. La varia natura dei messaggi, così come dei topic, rende necessario l'utilizzo di vari formati di comunicazione, da valori singoli a vettori di stringhe o valori numerici. Tutti i segnali, dalla variabile di controllo alla trasmissione di una telecamera, sono interpretati come messaggi pubblicati su un topic.

Per rendere più comprensibile lo sviluppo di un sistema complesso gruppi di nodi possono essere associati ad un **package**, una cartella dove sono salvati i file di codice associati alle corrette dipendenze (richiamando cioè librerie definite altrove, ovvero in altri package) e l'avvio del singolo nodo prevede di localizzarlo puntando al package corretto. All'avvio di ogni nodo ROS costruisce anche lo spazio dedicato ai topic in cui il nodo scriverà o leggerà i messaggi di suo interesse.

Infine, un sistema robotico complesso può richiedere l'avvio di molti nodi, e non è possibile avviarli singolarmente delegando all'operatore la digitazione su terminale di una lunga sequenza di comandi. ROS pertanto istituisce la convenzione del **launchfile,** un file di testo in formato XML in cui sono contenute le istruzioni per l'avvio sequenziale di tutti i nodi necessari al funzionamento del sistema.

2.3 Struttura del software e necessità di interfaccia

Il design del software del rover segue la suddivisione dei sottosistemi, virtualmente indipendenti, affiancati da nodi di controllo, in particolare delle comunicazioni verso i microcontrollori, e dai nodi di interfaccia con l'utente. I sottosistemi principali previsti per l'azionamento del rover sono i seguenti:

- Locomozione
- Braccio robotico
- Trivella per campionamenti

Ai sottosistemi principali si affiancano i seguenti deputati all'acquisizione di segnali dai sensori

- IMU, ultrasuoni e GPS
- Sistema di potenza
- Camere e stereovisione

Ciascun sottosistema è in genere governato da un nodo dedicato al controllo che interpreta i comandi forniti dal joystick e instrada i segnali per il comando ai relativi microcontrollori, nei quali un ulteriore nodo è ospitato. Il microcontrollore si interfaccia con la scheda elettronica e riceve opportuni segnali dai sensori utili per il controllo o per il pilotaggio da parte

dell'operatore. In questa fase non è stato ancora realizzato un software per il comportamento autonomo. Nella maggior parte dei casi i messaggi sono organizzati in vettori di numeri decimali a 32 bit, i quali sono interpretati dal nodo sotto opportune convenzioni. In alcuni casi speciali (per esempio la stereovisione) le tipologie di messaggi si adattano ad alcuni standard definiti da ROS.

Ogni singolo sottosistema è ben rappresentato in Fig. 2.1, dove i nodi sono rappresentati dagli ovali che comunicano tra loro mediante i topic, incorniciati da rettangoli a cui arrivano e da cui partono frecce, rappresentazione grafica dello scambio di informazioni.

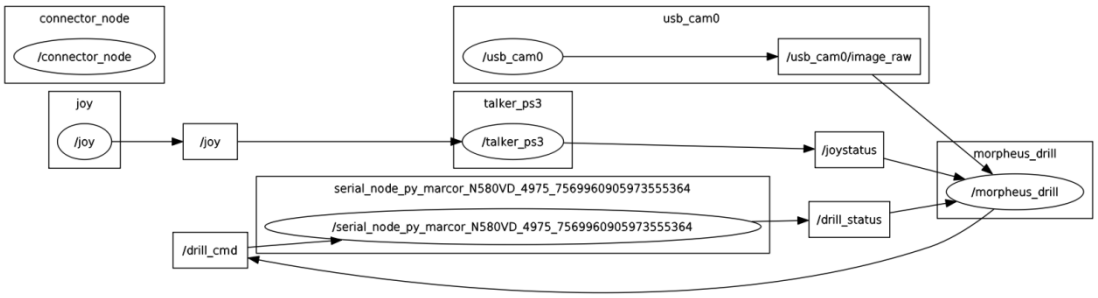


Figura 2.1. *Rappresentazione schematica dell'infrastruttura software del sottosistema "trivella", rappresentativa dei vari sottosistemi del rover*

L'esigenza di tradurre per l'operatore in maniera immediatamente comprensibile le informazioni utili estrapolandole dallo scambio di messaggi attraverso i topic ha portato all'esigenza della creazione di un'interfaccia grafica: ciò risolve la necessità di accedere mediante comandi convenzionali da terminale al contenuto dei singoli messaggi, in formato numerico senza legenda ed espresso secondo la convenzione decisa da uno dei programmatori dei singoli sottosistemi. In una singola schermata si sono dovuti riunire tutti le informazioni necessarie (scremando quelle ridondanti o inutili per l'operatore) per una visualizzazione e comprensione immediata. Tramite poi l'azione di bottoni o selettori, si possono espandere le possibilità di comando tramite joystick. Se il programmatore del sottosistema non coincide con l'operatore, l'interfaccia può garantire a quest'ultimo la non necessità di conoscere i comandi specifici di ROS per l'avvio dei nodi o per la visualizzazione del contenuto dei messaggi.

2.4 Introduzione al documento SRS

Come visto nel paragrafo precedente, il rover MORPHEUS presenta 3 principali sottosistemi differenti e un numero rilevante di sensori che sono impiegati non solo per fornire il *feedback* al codice di controllo degli attuatori ma anche per fornire informazioni sulla situazione del rover all'operatore. Seguendo le indicazioni dagli standard IEEE830 / IEEE29148 il primo passo nella progettazione consiste nel redigere un documento SRS – *Software Requirement Specification,*che includerà anche la lista completa dei requisiti e dei vincoli realizzativi.

In particolare il documento dovrà rispettare caratteristiche di completezza e di non ambiguità. Tutti i requisiti, sia funzionali che non, dovranno rispettare un'esigenza di tracciabilità: pertanto è richiesto che vengano contrassegnati da un indice univoco da richiamare nelle successive stesure della documentazione.

2.5 Documento SRS per l'interfaccia di MORPHEUS

1.1 Obiettivo

Lo scopo di questo documento è di fornire, per scopo informativo o di manutenzione del software, il quadro generale di come l'interfaccia grafica generale si colleghi al resto dell'infrastruttura software del rover MORPHEUS, di specificare quali siano i vincoli realizzativi (imposti dalla struttura pre-esistente o derivati da scelte di design) e di mostrare il collegamento tra gli elementi grafici a schermo e l'effettiva origine dei segnali a bordo del rover.

1.2 Campo d'applicazione

L'interfaccia sviluppata è stata creata *ad hoc* per MORPHEUS e punta principalmente ad un'essenziale interpretazione grafica dell'output dei sensori. L'interfaccia consente anche un input semplificato ed intuitivo di alcuni input di comando, per la maggior parte condensando la regolazione di parametri o l'invio di istruzioni nella pressione di singoli tasti da parte dell'operatore degli strumenti di input comuni (joystick, mouse e tastiera) piuttosto che con l'invio di stringhe di comando. Virtualmente, il prodotto consentirà in futuro il comando del rover anche ad operatori che non hanno partecipato allo sviluppo del software e che quindi non conoscono la grammatica e i comandi esatti per l'attivazione delle varie funzioni.

1.3 Definizioni, acronimi e abbreviazioni

Per esigenze di sintesi o perché di uso comune, alcuni acronimi verranno ripresi in questo documento e sono qui di seguito riportati.

IMU	Inertia Measurement Unit
GPS	Global Positioning System
GUI	Graphical User Inteface
ROS	Robot Operating System
USB	Universal Serial Bus

Per l'esigenza di non ambiguità si riportano qui alcuni termini, principalmente della struttura software di ROS, indispensabili per la definizione esatta dei requisiti (cfr. Par 3)

Nodo	Singolo script software eseguito in ROS
Topic	Spazio di memoria dinamico in ROS per la lettura/scrittura dei segnali
Package	Cartella di riferimento in ROS, in cui sono contenuti i files
RosParam	Parametro globale di funzionamento in ROS
LaunchFile	File che permette l'avvio sequenziale di più nodi
Publisher	Nodo che cambia i valori in un topic
Subscriber	Nodo che legge i valori di un topic
Qt / RQt	Libreria per la progettazione di GUI, e suo porting in ROS
OpenCV	Libreria per l'elaborazione di immagini
Python	Linguaggio di programmazione ad alto livello
C++	Linguaggio di programmazione ad alto livello

2 Descrizione generale

L'interfaccia si configura come nodo ROS eseguibile separatamente dagli altri nodi sviluppati per MORPHEUS deputati ai singoli sottosistemi; si configura inoltre come finestra eseguibile sia separatamente sia lanciata come plugin accanto ad altri proposti da ROS. Nel primo caso è sufficiente eseguire un launchfile che si occupa di attivare anche il joystick, la trasmissione delle telecamere ed eventuali altri nodi normalmente eseguiti in background durante ogni

accensione del rover. L'interfaccia è pensata per essere processata sul computer portatile (base station) deputato al controllo a distanza del rover, già configurato per interfacciarsi con i nodi ROS eseguiti nel computer di bordo del rover (Jetson TX1). Essendo in rover ancora in sviluppo, l'interfaccia è stata sviluppata in linguaggio Python (v. 2.7), adatto per la prototipazione rapida e per rapide modifiche in quanto interpretato e non necessitante di compilazione.

2.1 Inquadramento

Esistono numerosi plugin predefiniti del pacchetto ROS che consentono di controllare il cambiamento di valori in specifici topic, purchè se ne conosca nome, tipologia e valori di riferimento (minimo, massimo,etc.). L'interfaccia generale assolve alla funzione di controllo contemporanea di più elementi, organicamente disposti nella finestra, dell'interfaccia, potendo al tempo stesso lanciare istruzioni o gruppi di istruzioni personalizzate associate al semplice azionamento di elementi di controllo nella finestra di comando. L'interfaccia si avvale della libreria Qt nella sua versione Open Source (v. 4), e nel quale ambiente di sviluppo risulta facile l'aggiunta di elementi grafici aggiuntivi, oltre che essere pienamente compatibile con ROS (i plugin rqt si basano sulla medesima libreria) ed eseguibile sia in Python che in C++. L'unica eccezione riguarda le aree di visualizzazione grafica, che sono state sviluppate sfruttando la libreria aggiuntiva OpenCV (v. 2.4.x, usata anche da ROS). Tali problemi si sono presentati anche nel tentativo di includere il visualizzatore proprio di ROS (Rviz) all'interno dell'interfaccia: tale operazione sembra essere possibile solo mediante impiego di C++, ma il problema può essere aggirato affiancando le finestre eseguendole come plugin rqt)

2.2.1 Interfaccia sistema/utente

L'interfaccia si configura come una finestra principale nella quale comandi possono essere impartiti mediante clic del mouse (o associandoli alla pressione di specifici tasti della tastiera o del joystick). Il sistema è pensato per essere modulare ed eseguito dedicando anche finestre specifiche per un gruppo di operatori specializzati, eventualmente distribuendole su più schermi. Allo stato attuale, per ogni sottosistema principale del rover (Arm, Locomotion, Drill) si apre una finestra con i comandi dedicati al sottosistema; contestualmente il codice ROS per il controllo del sottosistema scelto viene avviato e vengono inibiti gli altri due.

2.2.2 Interfaccia hardware

Il sistema è eseguita senza rallentamenti su computer portatile con processore i7 7700HQ (anche per la rappresentazione di elementi grafici, attualmente non è ottimizzata per elaborazioni che sfruttino il chip grafico). Purchè siano rispettate i vincoli software, è immediatamente eseguibile o trasferibile su altri computer in comunicazione con il computer di bordo del rover, completo dei suoi sottosistemi.

2.2.3 Interfaccia software

Analogamente a tutto l'impianto del software di MORPHEUS, l'interfaccia si basa sull'utilizzo di ROS ("Indigo") in ambiente linux (Ubuntu 14.04.5 LTS "Trusty Tahr"). Come precedentemente descritto, l'interfaccia funziona sfruttando una serie di librerie interpretate in codice Python nella versione Python 2.7. Le altre librerie (la cui versione è determinata dai vincoli finora esposti) sono

Qt	versione 4	(libreria per interfaccia)
PyQt		(libreria per interpretazione di Qt in Python)
RQt		(libreria per intepretazione di Qt in ROS)
OpenCV	2.4.13	(libreria grafica open source, utilizzata anche da ROS)

Oltre a queste si rendono necessarie altre librerie standard di Python per attivare funzionalità avanzate del codice (os, sys, threading…) o librerie matematiche (math, numpy…) per l'interpretazione di segnali in formati numerici complessi o per lo svolgimento di operazioni di calcolo all'interno del codice.

2.2.4 Interfaccia di comunicazione

I driver per la comunicazione tramite mouse e tastiera sono normalmente implementati in ogni calcolatore tramite protocollo USB. Il driver per la comunicazione con il joystick si basa su ROS e attraverso un lavoro di mappatura preventivo può indifferentemente collegarsi con due dei joystick attualmente più comuni in commercio. La comunicazione tra base-station e computer di bordo è assunta come attiva e gestita tramite opportune configurazioni di ROS.

Ad opportuni nodi ROS è deputata la corretta attivazione e trasmissione dei dati da parte dei sensori.

2.2.5 Vincoli relativi all'occupazione di memoria

Nono sono stati posti vincoli, data la capacità del computer utilizzato come base station, nell'utilizzo dei supporti di memorizzazione e della memoria volatile (RAM).

2.2.6 Operazioni

Una volta avviato il computer di bordo del rover e stabilita la connessione con la base station, l'interfaccia si lancia digitando su terminale il comando

```
roslaunch my_gui gui_full.launch
```

è consigliabile ma non obbligatorio connettere il joystick prima dell'avvio dell'interfaccia.

2.2.7 Vincoli per installazione

Purchè si rispettino i vincoli illustrati nel punto 6.1.3 l'interfaccia è interpretata come un semplice package aggiuntivo all'interno dell'ambiente ROS. Per la lettura delle webcam ci si basa sul package aggiuntivo `usb_cam`.

2.3 Macro funzionalità del sistema

La prima interfaccia consente di avviare i nodi software dei vari sottosistemi abilitandone la funzionalità mediante la pressione dei tasti. Una stringa di testo comunica all'operatore l'effettiva esecuzione dei comandi mentre degli indicatori grafici traducono visivamente lo stato del rover derivato dai sensori. Un selettore numerico abilita la scelta di una delle camere di bordo e ne avvia la trasmissione proiettando l'immagine nell'area al centro della schermata.

L'attivazione dei bottoni (con il mouse o associata alla pressione di tasti) abilita o disabilita funzionalità dei vari sottosistemi, bloccando talvolta l'attivazione di funzionalità successive o concorrenti.

In basso a sinistra una serie di bottoni abilita il comando di uno specifico sottosistema (Arm, Locomotion, Drill) aprendo un'ulteriore finestra con interfaccia e funzionalità dedicata al singolo sottosistema. L'attivazione della finestra non coincide con l'attivazione del relativo nodo software, tale comando è deviato ad un pulsante di attivazione all'interno della finestra modale stessa.

Allo stesso modo un altro pulsante interrompe il comando del sottosistema, chiude la finestra modale e ritorna all'interfaccia principale.

Sottosistemi:

- ARM: nell'interfaccia del braccio sono presenti indicatori di estensione dei giunti (ulteriori indicatori possono essere inseriti in seguito) che riportano i dati dal feedback dei motori. Un tasto è dedicato a lanciare la funzionalità avanzata di ritorno in posizione chiusa

- LOCOMOTION: attorno ad uno schema del rover in pianta sono associati 6 indicatori testuali (uno per motore) che riportano l'intensità del comando erogato (o della velocità effettivamente raggiunta) al singolo motore, Tale valore è convertito anche in un'indicazione grafica con una spia che passa dal rosso (motore fermo) al verde (segnale massimo) passando per intensità graduali di giallo. Ulteriori colori codificati corrispondono a stati del motore definiti dal sistema di controllo

- DRILL: l'interfaccia include la schermata per il controllo visivo delle operazioni della trivella e degli indicatori di tensione applicata e corrente misurata nel circuito di alimentazione, oltre ad una stima della velocità. Con una stima visiva del livello di abbassamento della trivella, un selettore può impostare un livello approssimativo di discesa da raggiungere o comandare manualmente il movimento.

2.4 Caratteristiche degli utenti

Si presuppone che gli utenti raggiungano un certo grado di familiarità con l'interfaccia e memorizzino la mappatura dei comandi su tastiera/joystick; non si necessita da parte dell'operatore una conoscenza approfondita del codice di funzionamento dei vari sottosistemi né delle effettive caratteristiche degli attuatori comandati.

2.5 Vincoli generali

L'effettivo funzionamento efficace dell'interfaccia non può prescindere dal corretto funzionamento di ROS e dei nodi deputati al funzionamento del rover MORPHEUS. Il rover è ancora in fase di sviluppo al momento della stesura di questa SRS pertanto alcune delle funzionalità sono previste ma ancora non completamente operative.

2.6 Assunzioni e dipendenze (ipotesi iniziali)

La generazione dell'interfaccia prevede l'indicazione da parte dei singoli sviluppatori del team del numero e della tipologia dei messaggi instradati nei topic come input di comando o feedback dai motori, illustrati successivamente nei requisiti. Allo stato attuale la modifica da parte dello sviluppatore del team del metodo di comunicazione tra il nodo e il controllore dell'attuatore andrà ad annullare il funzionamento corretto dell'interfaccia. Si assume come già presente, testato e funzionante il collegamento tra i processi di ROS della base station e quelli a bordo del rover. Malfunzionamenti a bordo di qualsiasi genere possono essere segnalati sull'interfaccia ma non risolti se non intervenendo direttamente sul rover.

Si assume inoltre come valido il metodo *subprocess* di Python per lanciare comandi al terminale direttamente dallo script.

2.7 Requisiti da analizzare in futuro

Una volta terminato lo sviluppo del rover, l'interfaccia potrebbe essere migliorata sia in termini di tempi di esecuzione, di esecuzione e separazione del codice in file separati, arrivando a poter integrare un'interfaccia eseguibile senza conflitti su più terminali.

3 Specifiche funzionali e non funzionali del sistema

3.1 Requisiti funzionali

RF 1.1	**Interfaccia generale**	**Attivazione dei processi**
Descrizione	Autorizzazione al comando	
Input	Pressione dei primi tasti sull'interfaccia	
Sorgente segnale	Nessuna	
Processo	All'avvio dell'interfaccia tutti i tasti sono bloccati tranne quelli che permettono l'avvio della sequenza di operazioni Il sistema deve consentire l'avvio del rover in tre passaggi: o Accensione generale (avvio dei nodi centrali di controllo, lettura sensori) o Autorizzazione alle operazioni (pulsanti di "Stop" e "Go" mutuamente esclusivi). Solo in caso di pressione su "Go" il rover accetta l'avvio dei sottosistemi di attuazione o Selezione della modalità di comando (automatica o manuale) mutuamente esclusiva I tre passaggi devono avvenire nella sequenza specificata.	
Output	La pressione dei tasti è confermata graficamente, cambiando lo sfondo dei pulsanti con colori convenzionali (verde e rosso) La pressione dei tasti va a modificare opportuni parametri di controllo passati ai nodi software come parametri RosParam per permettere/inibire funzionalità	
TBD	La modalità automatica è prevista ma non ancora implementata	

RF 1.2	**Interfaccia generale**	**Carica delle batterie**
Descrizione	Indicatore della carica residua alle batterie del rover	
Input	Nessuno	
Sorgente segnale	Topic del sistema di alimentazione, singolo valore numerico	
Processo	All'avvio un indicatore fornisce una stima della carica residua delle batterie, sia con un valore percentuale sia tramite colori convenzionali. Degli avvisi vengono generati quando la carica scende sotto una certa soglia (60% e 30%)	
Output	Una barra colorata ha lunghezza proporzionale alla percentuale di carica residua, il colore ha transizioni proporzionali tra il verde (100%) e il rosso (0%) passando per giallo e arancione Degli avvisi testuali vengono generati quando la carica scende sotto una	

	certa soglia (60% e 30%)
TBD	Non è stato ancora implementato il controllo nel sistema di alimentazione

RF 1.3	**Interfaccia generale**	**Avvio lettura joystick**
Descrizione	Avvio manuale dei driver per la lettura del joystick collegato	
Input	Pressione del pulsante appropriato	
Sorgente segnale	Topic di interpretazione del joystick: joystatus. E' un vettore numerico di valori a virgola mobile a 32 bit.	
Processo	A seconda che il joystick collegato alla base station sia DualShock o XboxController la pressione del bottone lancia i driver per la lettura del segnale. I comandi sono stati mappati per poter funzionare indipendentemente dal joystick collegato	
Output	Conferma del pulsante di selezione premuto	
TBD		

RF 1.4	**Interfaccia generale**	**Attivazione dei sottosistemi**
Descrizione	Avvio manuale dei tre sottosistemi del rover	
Input	Pressione del pulsante appropriato	
Sorgente segnale	Nessuno	
Processo	Pressione del tasto per l'avvio di un singolo sottosistema e apertura relativa finestra modale. I tre sottosistemi sono mutuamente esclusivi	
Output	Conferma del pulsante di selezione premuto e lancio della finestra modale	
TBD		

RF 1.5	**Interfaccia generale**	**Indicatore d'assetto / ultrasuoni**
Descrizione	Indicatori grafici dell'assetto (roll/pitch) e di ostacoli rilevati dagli ultrasuoni	
Input	Nessuno	
Sorgente segnale	Nodo di lettura della scheda sensori: 4+2 valori numerici.	
Processo	Un applicativo per la resa grafica dell'assetto (roll/pitch) sfruttando gli elementi grafici di Qt è stato adattato e incorporato nell'interfaccia. 4 indicatori grafici corrispondenti ai sensori ad ultrasuoni (posizionati agli angoli del rover) si colorano con intensità proporzionale all'ostacolo rilevato dal sensore	
Output	Indicatori visivi	
TBD	La scheda sensori non è ancora provvista del nodo ROS per la trasmissione dei dati	

RF 1.6	Interfaccia generale	Visualizzazione telecamere
Descrizione	Incorporamento dell'immagine trasmessa da una delle telecamere di bordo	
Input	Selettore del numero di camera e tasto di avvio/arresto trasmissione	
Sorgente segnale	Nodo della telecamera del tipo `usb_cam/image_raw`, formato di ROS "image" interpretabile con OpenCV	
Processo	L'operatore seleziona in numero della camera desiderata e avvia la trasmissione con l'apposito tasto. Il medesimo tasto arresta la trasmissione	
Output	Nell'area centrale dell'interfaccia è generata l'immagine trasmessa dalle telecamera selezionata	
TBD	La larghezza di banda richiesta dalla trasmissione video	

RF 2.1	Interfaccia ARM	Operazioni sottosistema
Descrizione	Nella finestra modale sono compresi tutti i comandi di avvio delle operazioni per il braccio robotico e degli indicatori grafici	
Input	Pressione dei tasti nella finestra / comandi dal joystick	
Sorgente segnale	Nodo di comando del sottosistema (in caso di assenza di feedback) Nodo di controllo degli attuatori (TBD)	
Processo	L'operatore è in grado di abilitare /disabilitare le funzionalità del braccio mediante la pressione sui bottoni. Indicatori grafici mostrano la lettura di ritorno dagli attuatori (se disponibile) come livello percentuale. Un tasto è dedicato alla funzionalità di ritorno in posizione ritratta.	
Output	Valori di ritorno dall'attuatore	
TBD	E' in atto la riprogettazione del braccio e la conversione dei giunti da attuatori lineari a rotoidali. Potrebbe essere necessario ripensare la visualizzazione dei valori di ritorno (cfr. Cap. III)	

RF 2.2	Interfaccia LOCOMOTION	Operazioni sottosistema
Descrizione	Nella finestra modale sono compresi tutti i comandi di avvio delle operazioni per il sistema di locomotion e degli indicatori grafici	
Input	Pressione dei tasti nella finestra / comandi dal joystick	
Sorgente segnale	Nodo di comando del sottosistema (in caso di assenza di feedback) Nodo di controllo degli attuatori (array di 12 valori numerici)	
Processo	Contestualmente all'avvio della finestra modale viene abilitato il nodo di comando del sottosistema. L'interfaccia mostra schematicamente la disposizione dei 6 motori sul rover: a ciascuno è associato un valore numerico e un indicatore grafico. I sei motori, virtualmente indipendenti, sono attualmente programmati per lavorare in modalità skeed-steering, ovvero tutti i motori dallo stesso lato ricevono lo stesso comando. Il nodo di controllo di ciascun motore può essere rilevato in quattro stati diversi: • SETUP: Il nodo di controllo modifica i parametri di	

	funzionamento del motore. Nell'interfaccia l'indicatore grafico usa il colore convenzionale blu e l'etichetta testuale è fissa a zero. • CHECK STATUS: Il nodo di controllo attiva una routine di controllo interrompendo il normale funzonamento. Nell'interfaccia l'indicatore grafico usa il colore convenzionale azzurro e l'etichetta testuale è fissa a zero. • AZIONAMENTO: Il motore riceve un comando dall'operatore e si porta alla velocità selezionata. Nell'interfaccia la velocità misurata (oppure, se non presente nessun feedback, la velocità imposta dal comando) è riportata nell'etichetta testuale. L'indicatore grafico varia il suo colore da rosso (motore fermo) a verde (motore al massimo) con intensità proporzionale alla velocità comandata. • ERRORE: Il nodo di controllo non è più in comunicazione con il motore. L'indicatore grafico diventa nero ed è visualizzato un messaggio di errore. In aggiunta, un selettore permette di regolare la percentuale di potenza, rispetto al massimo fissato dal costruttore, utilizzabile dai motori.
Output	Valori di ritorno dai sei motori
TBD	In caso di necessità, l'interfaccia può essere riprogrammata parallelamente al nodo di comando per rendere i sei motori indipendenti.

RF 2.3	**Interfaccia DRILL**	**Operazioni sottosistema**
Descrizione	Nella finestra modale sono compresi tutti i comandi di avvio delle operazioni per la trivella e degli indicatori grafici	
Input	Pressione dei tasti nella finestra / comandi dal joystick	
Sorgente segnale	Nodo di comando del sottosistema (in caso di assenza di feedback) Nodo di controllo degli attuatori (array di 6 vettori numerici)	
Processo	L'operatore è in grado di abilitare /disabilitare le funzionalità della trivella mediante la pressione sui bottoni. La pressione sul bottone principale abilita tutti gli altri bottoni della finestra e avvia il nodo di comando della trivella. Al centro della finestra è mostrato il video della telecamera dedicata al monitoraggio del sottosistema. Oltre che con il joystick, la trivella può essere pilotata manualmente con una serie di pulsanti sulla sinistra per la salita/discesa del carrello e della velocità di rotazione. Indicatori testuali sulla sinistra mostrano i seguenti valori per entrambi i motori del sottosistema: • Tensione (imposta) • Corrente (misurata)	

	• Velocità di rotazione (calcolata)
Output	Immagine trasmessa dalla videocamera Valori di ritorno dalla scheda di controllo
TBD	

3.2 Requisiti non funzionali

RN 1	**Requisiti prestazionali**	**Eseguibilità**
L'interfaccia deve poter essere eseguita (anche in contemporanea ad altri processi) sul computer che funge da base station. La parte pesante dell'esecuzione (elaborazione degli indicatori grafici) non deve essere bloccante per il resto dell'interfaccia, al costo di un maggiore tempo di risposta. L'interfaccia non è caricata sul computer di bordo		

RN 2	**Vincoli generali del progetto**	**Vincoli software**
L'interfaccia deve essere pienamente compatibile con ROS e il resto del software del rover. Pertanto • Linguaggio di sviluppo: C++ oppure Python • Compatibilità con il plugin RQt di ROS (v. "Indigo") • Indipendenza dalla tipologia di joystick utilizzato		

RN 3	**Vincoli generali del progetto**	**Risoluzione video**
La risoluzione video per la visualizzazione delle telecamere deve poter essere modificabile: in caso di problemi di ampiezza di banda per la trasmissione, deve poter essere ridotta.		

RN 4	**Vincoli generali del progetto**	**Periferiche di input**
Input nell'interfaccia ammessi per l'operatore: • Tastiera • Mouse • Joystick L'assenza eventuale del mouse non deve ridurre le capacità dell'interfaccia		

RN 5	**Vincoli generali del progetto**	**Versatilità**
Essendo lo sviluppo dell'interfaccia dipendente dallo sviluppo del rover, essa deve poter essere modificata facilmente.		

CAP. III - CONTROLLO SEMPLIFICATO DEL BRACCIO ROBOTICO

3.1 Introduzione al modello virtuale

Dopo aver realizzato un'interfaccia grafica generale per il controllo globale del rover, il lavoro di tesi si è focalizzato sulla realizzazione di un metodo per il controllo semplificato del braccio robotico, sfruttando la stereocamera come metodo per la ricostruzione dello spazio circostante e la pianificazione della traiettoria. L'esigenza di pianificazione della traiettoria implementerebbe la capacità del braccio di interagire con il terreno, essendo il progetto finalizzato allo scavo e alla raccolta di campioni dal suolo.

Il primo passo consiste nell'implementare un modello a giunti mobili interpretabile da ROS, sfruttando il formato URDF (Unified Robot Description Format) che condensa in un file in linguaggio XML la descrizione dei vari elementi rigidi (link) che compongono il sistema, insieme ai giunti mobili a cui sono connessi. Un primo esempio di link potrebbe essere il seguente:

```
<?xml version="1.0"?>
<robot name="Robot 1">
  <link name="Link 1">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </visual>
  </link>
</robot>
```

I vari tag del formato XML definiscono la gerarchia dei parametri descrittivi del sistema: un cilindro di raggio 20 cm (l'unità di misura standard è il metro) e altezza 60 cm è il solido che compone il "Link 1" all'interno del "Robot 1". Il tag `<visual>` corrisponde alla parte visualizzabile del sistema, a cui in parallelo si può affiancare una parte fisica per la gestione delle collisioni, non per forza coincidente con la prima.

```
<?xml version="1.0"?>
<robot name="Robot 1">
  <link name="Link 1">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </collision>
  </link>
</robot>
```

Come si può vedere dall'esempio, altri tag permettono di definire varie proprietà del link, tra cui massa, matrice di inerzia, colore. Un modo più intelligente di gestire la descrizione di un sistema complesso , dove gli stessi parametri devono essere gestiti in parallelo in caso di modifiche, sfrutta delle macro definite nel linguaggio XML, dove variabili (e formule) possono essere definite univocamente. Per esempio si può uniformare il raggio del cilindro tramite l'istruzione

```
<xacro:property name="R" value="0.2" />
```

E richiamare la variabile R nelle definizioni delle geometrie, introducendo al contempo una relazione di proporzione tra le due dimensioni

```
  <link name="Link 1">
    <visual>
      <geometry>
        <cylinder length="${3 * R}" radius="${R}"/>
      </geometry>
      <material name="blue"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="${3 * R}" " radius="${R}"/>
      </geometry>
    </collision>
  </link>
```

Per completare il modello, si definiscono dei giunti che collegano i vari link, secondo il formato nell'esempio seguente:

```
<xacro:property name="alpha" value="1" />
<xacro:property name="pi" value="3.1415" />
  <joint name="Joint 1_2" type="revolute">
    <axis xyz="1 0 0"/>
    <limit effort="10.0" lower="0.0" upper="${pi/2}" velocity="0.5"/>
    <origin rpy="0 ${alpha}"  0" xyz="${3 * R} 0 0"/>
    <parent link="Link 1"/>
    <child link="Link 2"/>
  </joint>
```

Il giunto così descritto unisce il Link 1 al Link 2, permettendo a quest'ultimo di ruotare attorno all'asse X in accordo con il versore definito associato al tag `axis`. Tale giunto è posizionato rispetto al sistema di riferimento del giunto precedente, assunto come origine, traslato di 60 cm lungo l'asse x e poi ruotato di 1 radiante attorno all'asse Y. La quantità `rpy` sta proprio ad indicare una rotazione secondo la sequenza convenzionale roll-picth-yaw definita rispetto agli assi fissi del sistema di riferimento precedente. Si noti che anche il valore convenzionale π deve essere definito a priori. Ciò è alla base del sistema di trasformazione di coordinate integrato con ROS, una libreria denominata "TF". Esso sintetizza i sei parametri necessari a comporre la matrice di trasformazione (ovvero roto-traslazione) privilegiando una scrittura più sintetica rispetto alla forma algebrica utile per i calcoli del tipo

$$ {}^{1}_{2}T = [{}^{1}_{2}R|\ {}^{1}t_{2}] \tag{3.1} $$

descrivendo la matrice di trasformazione del sistema 2 rispetto al sistema 1. Con riferimento all'esempio del giunto precedente, in coordinate omogenee, la matrice di trasformazione del link 2 rispetto al link 1:

$$ {}^{1}_{2}T = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 3R \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

Come si può osservare, la matrice così definita è utile per trasformazioni puramente geometriche, mentre nel caso del robot si osserva che il formato URDF deve poter essere in grado di gestire l'andamento nel tempo del braccio, seguendo le variazioni dei giunti rotoidali; sono inoltre imposti dei vincoli di estensione angolare (differenti per i due versi di rotazione) e anche dei limiti di coppia e velocità dipendenti dal tempo. La libreria TF risolve implicitamente il problema con la struttura Publisher-Subscriber di ROS: i valori delle variabili per le trasformazioni sono scambiati tramite topic (uno per giunto) e associati ad un

indice temporale. In questo modo è possibile introdurre la seguente relazione tra due sistemi di riferimento in istanti diversi

$$ {}^{b@t1}_{a@t0}T = {}^{b@t0}_{a@t0}T * {}^{b@t1}_{b@t0}T \tag{3.3} $$

Infine, la libreria TF è compatibile con il visualizzatore grafico di ROS, per cui risulta possibile generare un primo modello fisico del braccio robotico di MORPHEUS.

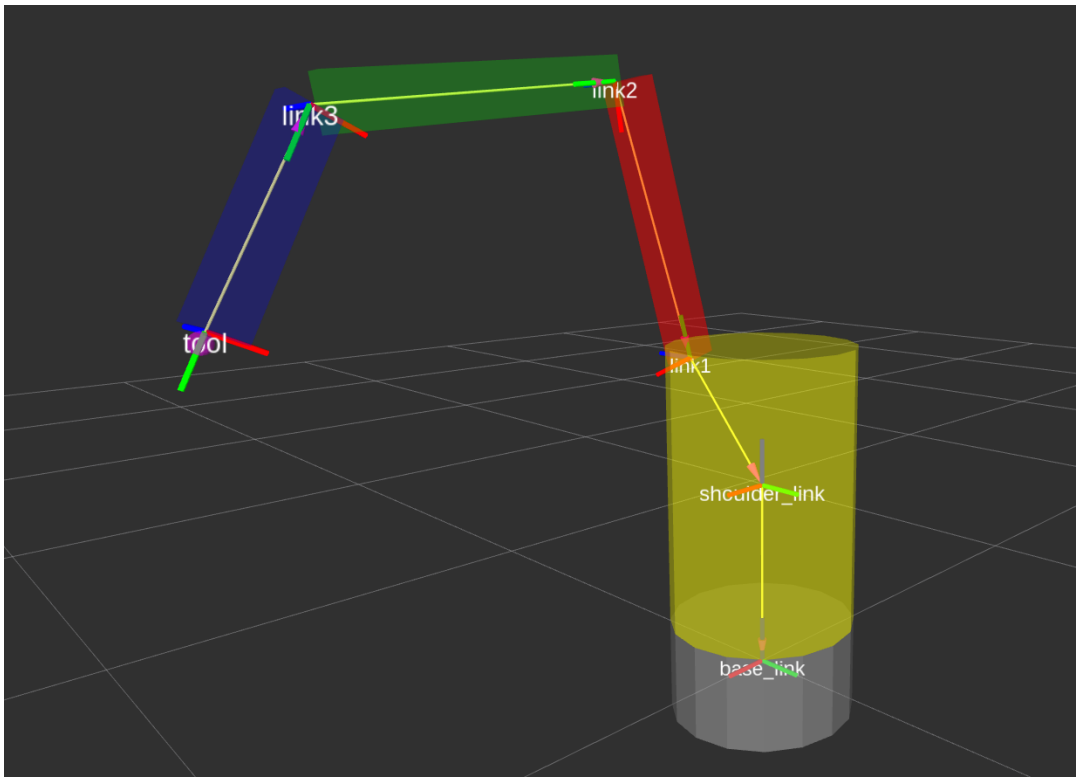


Figura 3.1. *Modello della struttura cinematica del braccio robotico in ambiente virtuale. I link sono rappresentati dalle geometrie solide, i giunti dai sistemi di assi orientati*

Nell'ambiente virtuale, ogni giunto è rappresentato da un solido, diverso nel colore per chiarezza, dove il cilindro grigio rappresenta il telaio posizionato nell'origine del sistema di riferimento. Tutti i giunti, rappresentati con i tre assi che seguono la sequenza RGB per la colorazione, sono posizionati in modo tale che la rotazione avvenga sempre attorno all'asse Z (blu) del rispettivo giunto. Variando il valore corretto nel topic `/joint_states` generato contestualmente al modello, un doppio vettore contenente il nome e l'istantaneo valore di rotazione degli angoli rispetto alla posizione di base, il visualizzatore aggiorna in tempo reale la posizione di ogni giunto. Il modello così costruito, anche se cinematicamente rappresentativo del reale braccio robotico in progettazione per il rover, non è però sufficientemente realistico per la pianificazione della traiettoria. In più l'ambiente virtuale,

senza altre informazioni, non rispecchia ancora lo scenario operativo nel quale il braccio si dovrà muovere.

3.2 Modello del braccio reale in ambiente virtuale

Partendo dal modello semplificato mostrato nel paragrafo precedente, si è cercato il modo di trasferire una geometria quanto più somigliante possibile al progetto meccanico del braccio. Partendo dalle tavole dei singoli pezzi del progetto, è possibile riunire tutte le parti che compongono un singolo link e generare una mesh che definisca il volume esterno dell'assieme. Rispetto al sistema di riferimento originale, si è dovuto spostare il sistema di riferimento di ogni raggruppamento nella posizione quale lo si desidera, con l'origine posizionata nel punto di movimento del giunto e l'asse Z parallelo all'asse di rotazione finale, passando dal formato di Solidworks al formato IGES, quindi esportando nel formato STL, la mesh può essere accettata e importata all'interno del formato URDF al posto di geometrie predefinite, mediante l'istruzione del tipo:

```
<visual>
  <geometry>
    <mesh filename="package://virtual_arm/meshes/link1.stl">
  </geometry>
  <material name="red"/>
</visual>
```

che prende in ingresso il file STL posizionato in una opportuna sottocartella del package sviluppato per questo lavoro. Grazie al lavoro preparatorio descritto prima, nel file URDF le traslazioni dell'origini di un giunto rispetto al precedente vanno quindi a coincidere con le lunghezze geometriche del giunto (già allineate agli assi principali) mentre si considera il primo asse di rotazione Z "verticale", mentre gli assi di rotazione degli altri giunti risultano paralleli al netto di una trasformazione di *roll-pitch-yaw* corrispondente ai valori [$\pi/2$ 0 $\pi/2$] (espressi in radianti) tra il primo link e la piattaforma di rotazione ancorata al telaio del rover. Un'ulteriore rotazione di π è applicata come *yaw* ai giunti successivi per permettere un'estensione del braccio applicando solo valori positivi come angoli di giunto. Il file che definisce il braccio reale, riportato in Appendice A, porta al primo risultato di valore: la costruzione del braccio in ambiente simulato. Poiché al braccio manca ancora un manipolatore, è stato inserito un generico *tool* cilindrico di forma arbitraria, rappresentato dal colore viola.

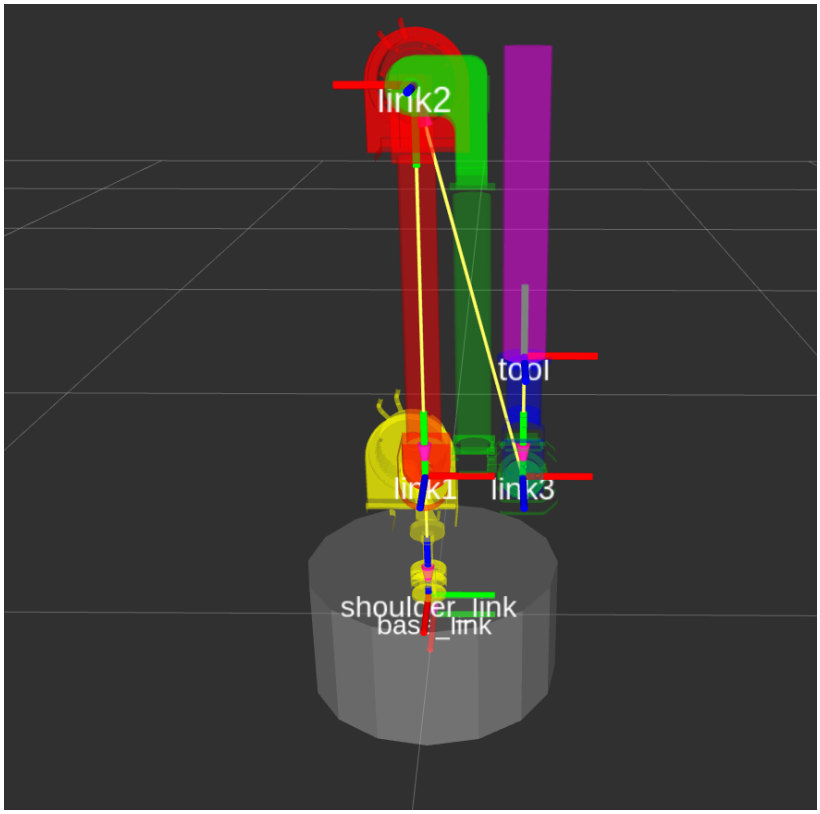
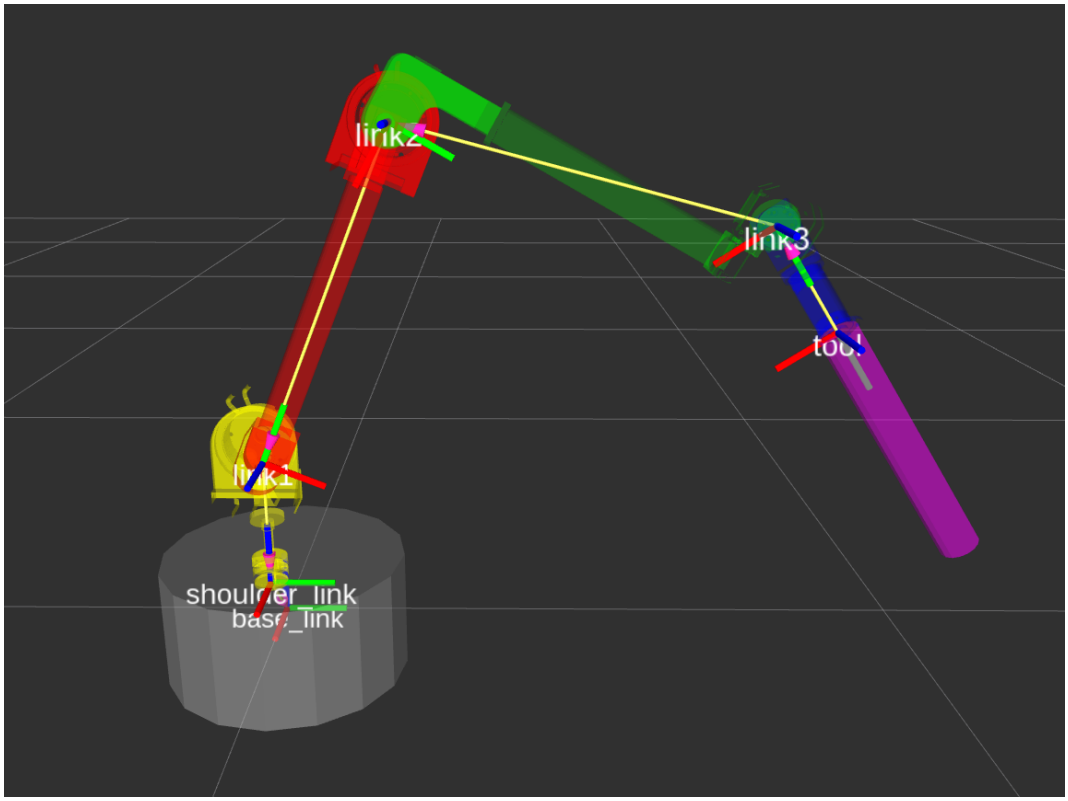


Figura 3.2 *Modello del braccio in ambiente virtuale in configurazione retratta (sopra) e in una configurazione con giunti estesi (sotto).*



Questo modello del braccio, che può essere collegato e animato tramite comando del joystick per il raggiungimento della posizione desiderata nei limiti dei vincoli definiti dal modello, necessita ora di un algoritmo di pianificazione della traiettoria, tale che possa effettuare in maniera immediata la sintesi dei valori istantanei dei vari giunti che permettano il passaggio del braccio dalla posizione A alla posizione B, definite dall'operatore, in maniera continua. Inoltre non risulta di nessuna utilità la movimentazione del braccio tra due posizioni arbitrarie, se esse sono completamente svincolate dall'ambiente operativo. Gli obiettivi del braccio robotico sono riassumibili nelle seguenti funzionalità, che necessitano di conoscere la posizione del rover rispetto all'ambiente:

- Scavo del terreno
- Raccolta di campioni e deposito in contenitori a bordo del rover

Bisogna quindi ricostruire la scena nell'ambiente virtuale, e ciò può essere fatto tramite la stereocamera integrata a bordo del rover. Apposite librerie software, mettendo in relazione punti omologhi nelle immagini delle due telecamere destra e sinistra, possono tramite una relazione geometrica di triangolazione ricavare le coordinate tridimensionali di alcuni punti inquadrati, permettendo di generare una nuvola di punti (associando anche il colore) e permettendo una ricostruzione della scena. Le coordinate di tali punti sono ovviamente riferite al sistema di riferimento di una delle due telecamere. Sono numerosi i metodi di applicazione pratica della triangolazione, non riportati nel dettaglio in questa tesi; inoltre le librerie di ROS prediligono l'applicazione del metodo della disparità per la creazione di una nuvola di punti "densa".

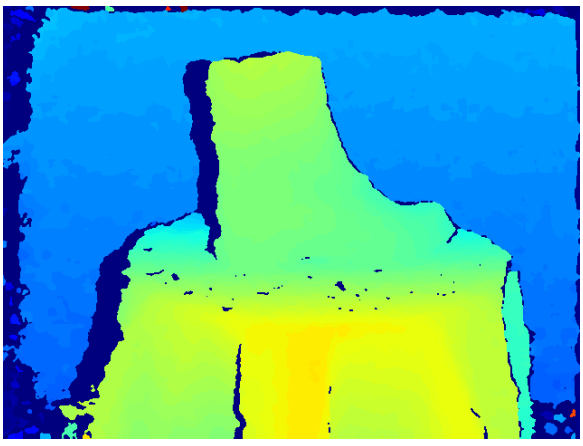


Figura 3.4 (sinistra) *Immagine tratta dalla camera sinistra di una stereocamera*
Figura 3.5 (destra) *Disparità calcolata relativa all'oggetto nell'immagine precedente. I giallo rappresenta valori di disparità più alti (relativi a punti più vicini alla stereocamera) mentre il blu rappresenta valori di disparità tendenti a zero.*

Si definisce disparità il raffronto (su una base di ottimizzazione numerica) delle due immagini rettificate per determinare il corretto spostamento di ciascun pixel (rispetto all'area che lo circonda) nell'immagine destra rispetto al suo omologo nella sinistra, per poi correlare la distanza del punto dalla telecamere in maniera inversamente proporzionale alla disparità calcolata.

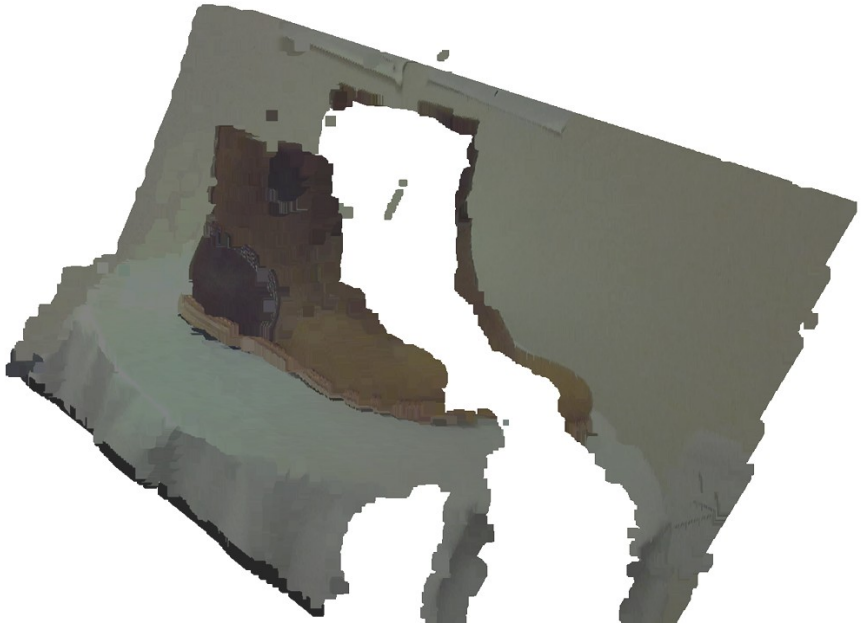


Figura 3.6 *Nuvola di punti tridimensionali ricostruita a partire dalla disparità di Fig. 3.5*

Una volta impostata la ricostruzione della nuvola di punti, essa all'interno di ROS è un tipo di messaggio standard, pubblicato su topic, e può essere importata nell'ambiente virtuale una volta determinata la trasformazione dal sistema di riferimento della camera a quello del telaio. Una prima stima, dedotta dalle tavole di costruzione, potrà essere successivamente raffinata mappando posizioni note del braccio all'interno della scena 3D.

3.3 Pianificazione semplificata della traiettoria

Il pacchetto software selezionato per la pianificazione di traiettoria è un plugin di ROS denominato "MoveIt!". L'obiettivo è generare un opportuno file di configurazione che interpreti correttamente il modello, ne gestisca correttamente le collisioni tra le proprie parti e introduca un cursore selezionabile sul punto di azione del end-effector. All'interno della scena l'operatore può quindi spostare l'end-effector da una posizione di partenza ad una di arrivo e

valutare la soluzione di cinematica inversa migliore che evolve il braccio dalla configurazione iniziale a quella finale. Nel pacchetto sono disponibili quattro librerie di algoritmi per la soluzione. Nella configurazione di prova verrà valutata la libreria Kinematics and Dynamics Library (KDL). La configurazione dei file prevede quindi la generazione di una serie di file di configurazione che contengono i parametri scelti per la soluzione, e una serie di launchfiles che istanziano il visualizzatore dell'ambiente virtuale e il collegamento con un topic `/joint_states` finalizzato al movimento del modello.

La simulazione, in realtà virtuale, può essere estesa in realtà aumentata sovrapponendo il modello (o una sua schematizzazione) e la traiettoria percorsa direttamente nella schermata della telecamera: ciò è possibile perchè sono note, grazie alle trasformazioni, le posizioni dei giunti rispetto al sistema di riferimento della telecamera. Note le carratteristiche di calibrazione della telecamera si possono convertire direttamente le coordinate 3D delle posizioni dei giunti (e di ogni punto del meccanismo) in corrispodenti coordinate nel piano immagine sfruttando l'equazione di proiezione che deriva dal modello matematico della camera.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.4}$$

Con la stessa libreria OpenCV sfruttata per l'elaborazione e la visualizzazione del segnale della telecamera nell'interfaccia, è quindi possibile relizzare una schematizzazione della configurazione del braccio in ogni istante tramite la funzione *cv2.projectPoints* che impelementa in maniera diretta ed efficiente la formula sopracitata. Includendo il tutto nella struttura di ROS, la configurazione dei giunti viene aggiornata in tempo reale seguendo le variazioni del topic appropriato.

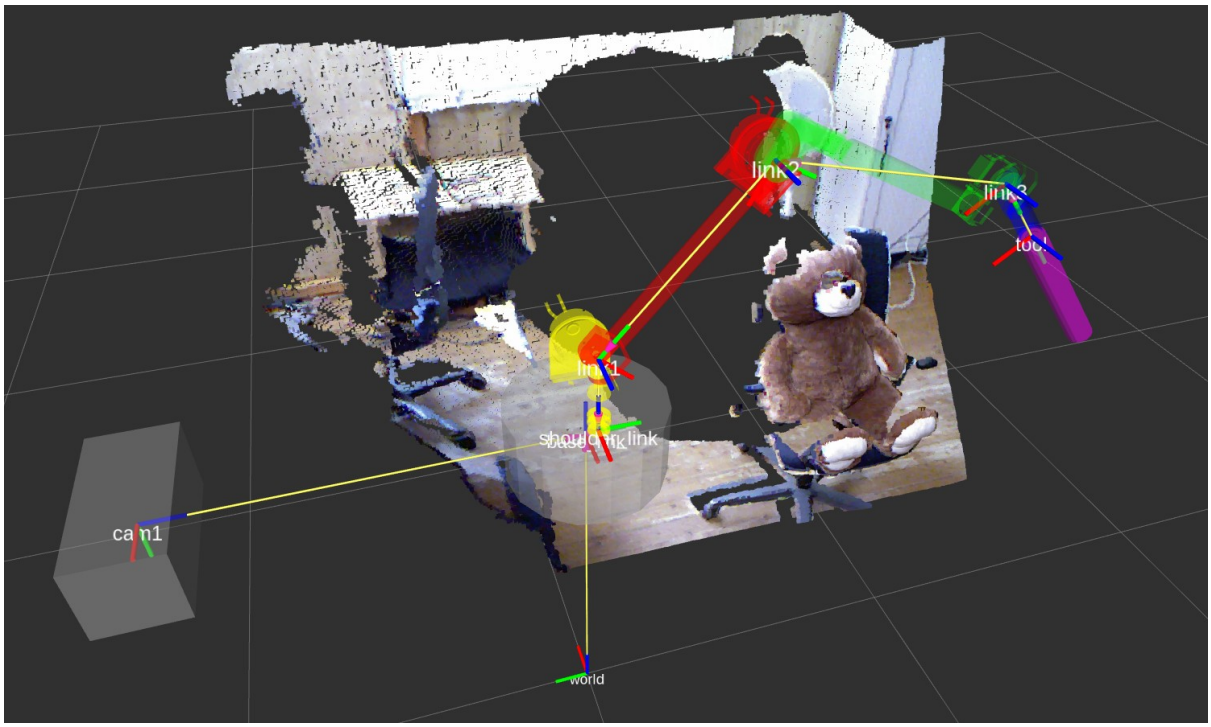


Figura 3.7 *Modello virtuale del braccio e nuvola di punti collegati allo stesso sistema di riferimento e visualizzati in contemporanea nell'ambiente virtuale. Il dataset della nuvola di punti è stato scaricato da https://vision.in.tum.de/data/datasets/rgbd-dataset/download*

Come si può vedere dalla figura, il modello del braccio è integrato nello scenario misurato e ricostruito in tre dimensioni grazie alla stereocamera. Nell'ambiente virtuale, sarà sufficiente che l'operatore trascini con il mouse (oppure aggiusti tramite joystick) il tool nella nell'orientamento desiderato rispetto al punto di interesse. Attivando la pianificazione di traiettoria, un'animazione mostrerà il movimento del braccio articolato rendendo possibile la verifica del movimento rispetto all'ambiente.

La nuvola di punti, ricavata dalla stereocamera, non presente nella figura, non è ovviamente esente da disturbi quali l'occlusione (il pupazzo molto vicino alla stereocamera copre una grande porzione di oggetti sullo sfondo).

CAP. IV - RISULTATI RAGGIUNTI E SVILUPPI FUTURI

Lo sviluppo dell'interfaccia per MORPHEUS, svolto in parallelo al lavoro sugli altri sottosistemi, è pronto per consentire in linea teorica una fase sperimentale di comando del rover. Tuttavia, in nessuno dei sottosistemi è stata completata la componente elettromeccanica, in alcuni casi ancora in fase di progetto. Operativamente, l'interfaccia è stata sperimentata separatamente sui tre sottosistemi di attuazione principale (locomotion, braccio e trivella) con i seguenti risultati

4.1 Interfaccia generale

L'interfaccia interagisce corriettamente dalla base station con i nodi di controllo nel computer di bordo del rover. Gli indicatori grafici sono stati provati con segnali simulati e sono stati risolti tutti i problemi di rallentamento dovuti all'aggiornamento della grafica. L'inclusione della trasmissione della telecamera (selezionando quella di interesse) è stata verificata ma non in contemporanea all'invio di altri comandi tra rover e base station.

Lo sviluppo dell'ambiente virtuale è visualizzabile in una finestra affiancabile all'interfaccia (sono entrambi sviluppati come plugin di rqt) ma non sarà possibile includerlo fino a che il codice dell'interfaccia non sarà riscritto in C++

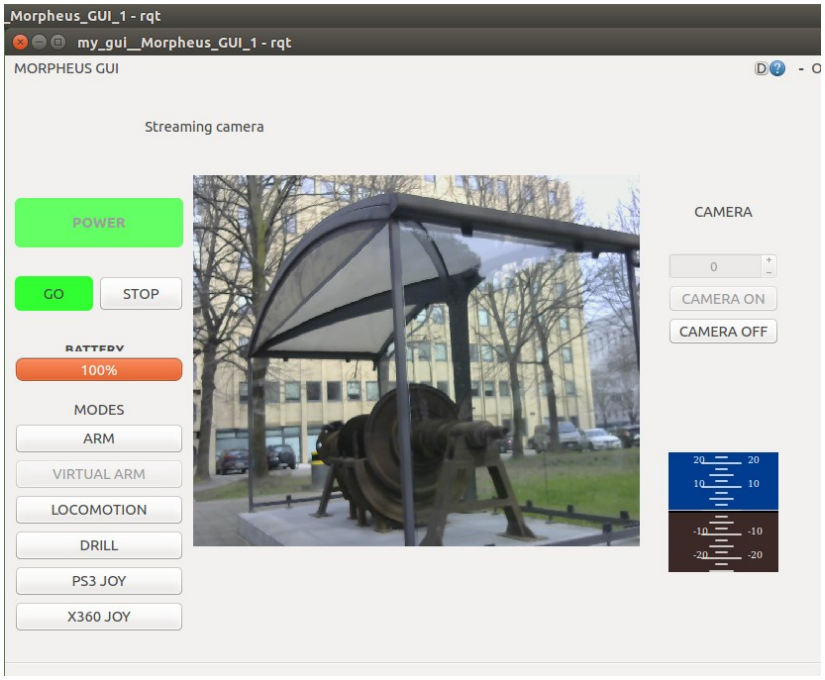
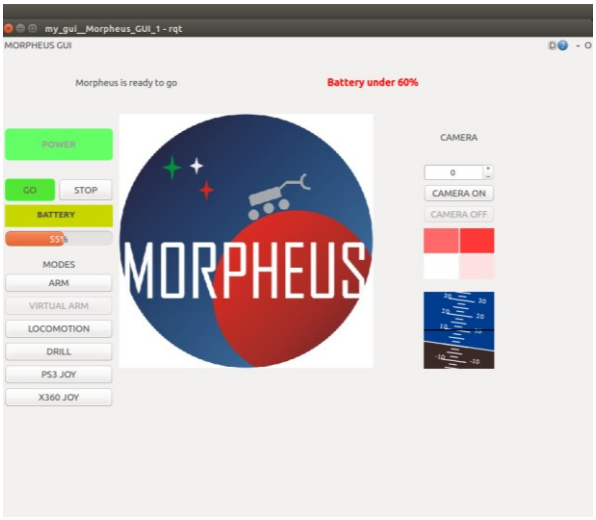


Figura 4.1 (sopra) *Finestra modale dell'interfaccia generale. Include la trasmissione della videocamera*

Figura 4.2 (sotto) *Finestra modale dell'interfaccia generale. Sono visibili l'attivazione degli indicatori relativi alla batteria sulla destra, un indicatore grafico per i sensori ad ultrasuoni e la visualizzazione di un segnale d'assetto*



.

4.2 Locomotion

E' il sottosistema più completo finora, benchè l'elettronica di bordo non consenta ancora di ricevere feedback dai motori. Nei test sul campo mappa i segnali inviati dal nodo di controllo e riceve correttamente le comunicazioni di stato dei microncontrollori. L'interfaccia non ha creato conflitti con il sottosistema nè ha indotto rallentamenti. Permette all'operatore di scegliere la potenza massima da erogare ai motori.

In Fig. 4.3 si nota come i due motori del lato sinisto ricevano un segnale con intensità massima rispetto alla potenza inviabile e si colorano di verde, mentre due motori del lato destro non ricevono un comando e mantengono l'indicatore grafico rosso. I due motori posteriori, disabilitati, sono visualizzati con un codice colore diverso.

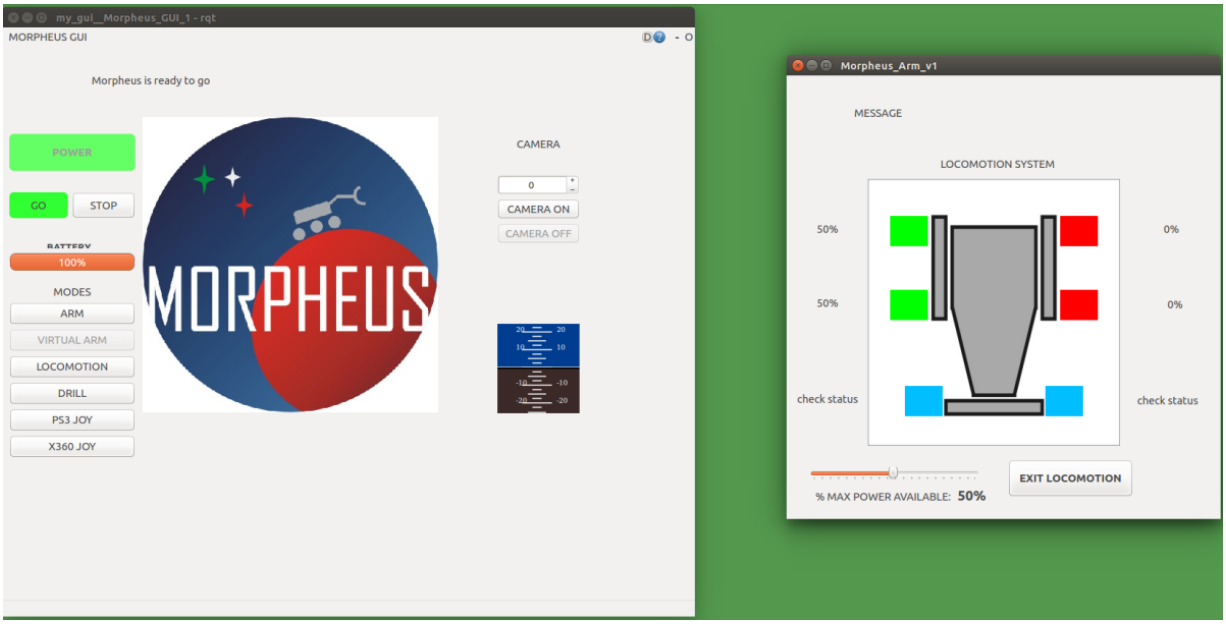


Figura 4.3 *La finestra modale del sottosistema locomotion affianca l'interfaccia generale*

4.3 Braccio

Il braccio è il sottosistema in fase di riprogettazione e pertanto non è stato possibile eseguire alcuna prova al di fuori dell'ambiente virtuale. Nella precedente versione l'interfaccia risultava in grado di avviare e disabilitare le funzionalità del sottosistema, tra cui il pulsante per la ritrazione automatica e la logica di controllo per la verifica dell'avvenuta attuazione. Nell'attuale iterazione di progetto, è stata appurata la procedura per importare le componenti dall'ambiente di sviluppo Solidworks e la definizione delle trasformazioni tra i giunti, parametrizzate per facilitare eventuali correzioni.

Nell'attuale configurazione quattro indicatori (sia grafici che testuali) sono predisposti per visualizzare il valore angolare (o di estensione) di ogni giunto.

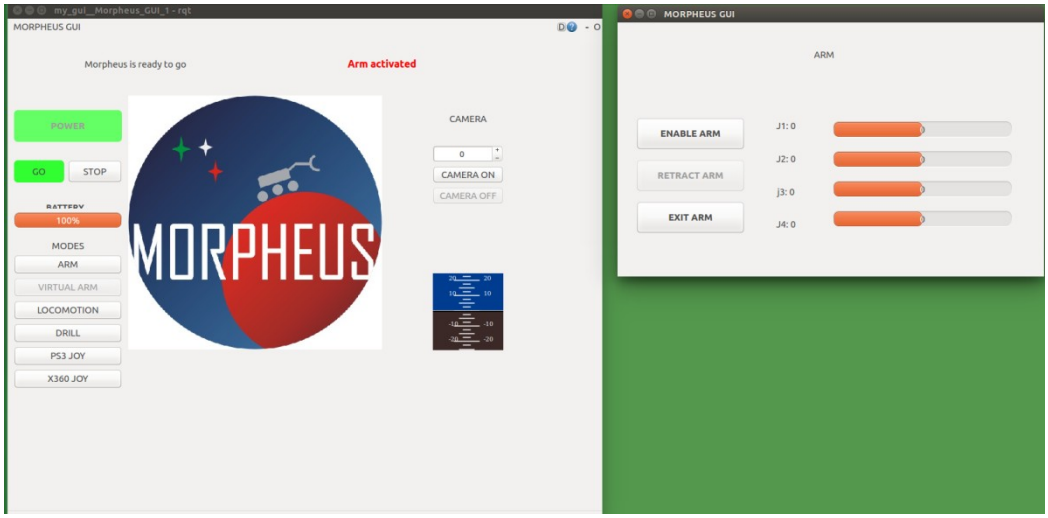


Figura 4.4 *Finestra modale dell'interfaccia per il braccio*

4.4 Trivella

L'interfaccia con la trivella (anch'essa ferma alla prima iterazione di sviluppo) è stata testata e risulta funzionante, comunicando correttamente con i microcontollori sia per l'invio di comandi sia per la ricezione delle misure dei valori elettrici nell'elettronica. E' predisposta per poter consentire all'operatore di scegliere il livello di discesa e la potenza massima di rotazione della trivella, con uno spazio per l'inquadratura di una telecamera di controllo.

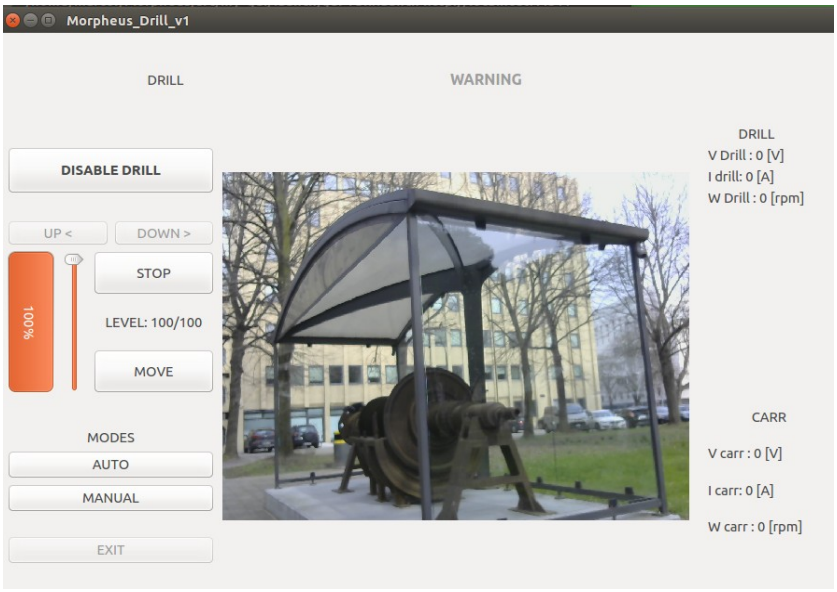


Figura 4.5 *Finestra modale dell'interfaccia per la trivella*

4.5 Sviluppi futuri

La progettazione modulare dell'interfaccia consente di poter modificare su indicazione dei progettisti dei singoli nodi software (o dell'operatore) le interfacce specifiche dei sottosistemi senza incidere minimamente sul resto del pacchetto. Con modifiche minime in futuro si potrà configurare l'interfaccia perchè sia possibile avviarla da computer diversi in rete, ognuno con un operatore specializzato.

Le interfacce sviluppate per questo progetto sono operative ma sviluppate con una logica di prototipazione rapida. Il primo sviluppo, una volta consolidate le esigenze di tutti i sottosistemi consisterà nella riscrittura completa del codice, suddividendolo in sottolibrerie da richiamare con una logica ad oggetti per una manutenzione più rapida, passando al contempo al linguaggio C++ per migliorare l'efficienza di esecuzione e consentire di fondere nello stessa finestra anche la parte di ambiente virtuale. Il linguaggio C++ è nativamente compatibile con libreria Qt scelta per la creazione dell'interfaccia.

Nell'ambito della pianificazione di traiettoria, una volta completata la realizzazione del braccio dal punto di vista elettromeccanico, una fase di prove consentirà di calcolare effettivamente la posizione del braccio nel sistema di riferimento della telecamera e verificare l'attinenza tra le configurazioni calcolate in realtà virtuale e quelle in ambito operativo. Si potrà includere il trasferimento delle informazioni in ambiente virtuale direttamente sovrapponendole alla trasmissione da telecamera, e consentendo direttamente da lì all'operatore di identificare o selezionare elementi di interesse nello scenario.

Indice degli acronimi

GPS - Global Positioning System

GUI - Graphical User Interface

IEEE - Institute of Electrical and Electronic Engineers

IGES - Initial Graphics Exchange Specification

IMU - Inertial Measurement Unit

KDL - Kinematics and Dynamics Library

LAN - Local Area Network

P2P - Peer-to-peer

ROS - Robot Operating System

SRS - Software Requirements Specification

STL - STereo Lithography interface format

TBD - To Be Determined

TF2: 2nd generation TransForm Library

URDF - Universal Robot Description Format

XML - eXtensible Markup Language

Bibliografia

Asad, Y., W. Lehman, M. A. Mustafa, M. M. Hayder (2015) Introducing Kinematics with Robot Operating System (ROS); *122nd ASEE Annual Conference and Exposition*, Paper ID # 11460

Cheng Y., Maimone M. W., Matthies L., (2006) Visual Odometry on the Mars Exploration Rovers; *IEEE Robotics & Automation Magazine*

Chitta S., I. A. Sucan, S. Cousins (2012) Moveit![ROS topics]; articolo in IEEE Robotics & Automation Magazine

Craig, J. J. (1986) Introduction to Robotics: Mechanics and Control (Pearson)

Foote T., (2013) tf: The Transform Library; Technologies for Practical Robot Applications (TePRA) In IEEE International Conference

Paganini D., (2017) MORPHEUS: ideazione e progettazione del rover di Ateneo.

Quigley, M., B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng. (2009) ROS: an open-source Robot Operating System; *ICRA Workshop on Open Source Software*

Tripp, L. L. et al., (1998) IEEE Recommended Practice for Software Requirements Specifications; *IEEE Std 830-1998*

(2011) Systems and software engineering - Life cycle processes - Requirements Engineering; *ISO/IEC/IEEE 29148-2011*

Sitografia

https://docs.opencv.org/2.4.13/

http://doc.qt.io/

https://github.com/PickNikRobotics/rviz_visual_tools/tree/indigo-devel

https://github.com/ros/geometry2

https://github.com/ros-visualization/rqt

https://wiki.python.org/moin/PyQt

http://wiki.ros.org/it

APPENDICE A: file finale di configurazione del braccio

```
<?xml version="1.0"?>
<robot name="MORPHEUS_ARMv1" xmlns:xacro="http://ros.org/wiki/xacro">

  <xacro:property name="W" value="0.04" />
  <xacro:property name="H" value="0.1" />
  <xacro:property name="L1" value="0.537" />
  <xacro:property name="L2" value="0.537" />
  <xacro:property name="L3" value="0.183" />
  <xacro:property name="G" value="0.07" />
  <xacro:property name="R" value="0.2" />
  <xacro:property name="T" value="0.4" />
  <xacro:property name="Tr" value="0.03" />
  <xacro:property name="h" value="0.03" />
  <xacro:property name="r" value="0.02" />
  <xacro:property name="bodylen" value="0.2" />
  <xacro:property name="baselen" value="0.4" />
  <xacro:property name="wheeldiam" value="0.07" />
  <xacro:property name="pi" value="3.1415" />

  <material name="blue">
    <color rgba="0 0 1 0.5"/>
  </material>

  <material name="red">
    <color rgba="1 0 0 1"/>
  </material>

  <material name="green">
    <color rgba="0 1 0 0.5"/>
  </material>

  <material name="white">
    <color rgba="1 1 1 1"/>
  </material>

  <material name="yellow">
    <color rgba="1 1 0 1"/>
  </material>

  <material name="purple">
    <color rgba="1 0 1 1"/>
  </material>

  <material name="grey">
    <color rgba="0.5 0.5 0.5 1"/>
  </material>

  <link name="base_link">
    <visual>
      <geometry>
        <cylinder radius="${R}" length="${bodylen}"/>
      </geometry>
      <material name="grey"/>
    </visual>
```

```
  <collision>
    <geometry>
      <cylinder radius="${R}" length="${bodylen}"/>
    </geometry>
  </collision>
  <xacro:default_inertial mass="1"/>
</link>

<joint name="baser_to_shoulder" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort="1000.0" lower="-3" upper="3" velocity="0.5"/>
  <parent link="base_link"/>
  <child link="shoulder_link"/>
  <origin xyz="${R/2} 0 ${0.5*bodylen}"/>
</joint>

<link name="shoulder_link">
  <visual>
    <geometry>
      <mesh filename="package://virtual_arm/meshes/Assieme0_Metri.stl"/>
    </geometry>
    <material name="yellow"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://virtual_arm/meshes/Assieme0_Metri.stl"/>
    </geometry>
  </collision>
  <xacro:default_inertial mass="1"/>
</link>


<joint name="shoulder_to_link1" type="revolute">
  <axis xyz="0 0 1" />
  <limit effort="1000.0" lower="-3" upper="3" velocity="0.5"/>
  <parent link="shoulder_link"/>
  <child link="link1"/>
  <origin xyz="0 0 0.2" rpy=" ${pi/2} 0 ${pi/2}" />
</joint>



<link name="link1">
  <visual>
    <geometry>
      <mesh filename="package://virtual_arm/meshes/Assieme1_Metri.STL"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <material name="red"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://virtual_arm/meshes/Assieme1_Metri.STL"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </collision>
  <inertial>
```

```
      <mass value="1"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
  </link>

  <joint name="link1_to_link2" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="1000.0" lower="-4" upper="4" velocity="0.5"/>
    <parent link="link1"/>
    <child link="link2"/>
    <origin xyz="0 ${L1} 0" rpy="0 0 ${pi}"/>
  </joint>

  <link name="link2">
    <visual>
      <geometry>
        <mesh filename="package://virtual_arm/meshes/Assieme2_Metri.STL"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <material name="green"/>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://virtual_arm/meshes/Assieme2_Metri.STL"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/>
    </collision>
    <inertial>
      <mass value="1"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
  </link>

  <joint name="link2_to_link3" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="1000.0" lower="-4" upper="4" velocity="0.5"/>
    <parent link="link2"/>
    <child link="link3"/>
    <origin xyz="${-2*G} ${L2} 0" rpy="0 0 ${pi}"/>
  </joint>

  <link name="link3">
    <visual>
      <geometry>
        <mesh filename="package://virtual_arm/meshes/Assieme3_Metri.stl"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 ${h}"/>
      <material name="blue"/>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://virtual_arm/meshes/Assieme3_Metri.stl"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 ${h}"/>
    </collision>
```

```
    <inertial>
      <mass value="1"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
  </link>

  <joint name="link3_to_tool" type="fixed">
    <axis xyz="0 0 1"/>
    <parent link="link3"/>
    <child link="tool"/>
    <origin xyz="0 ${L3} 0"/>
  </joint>

  <link name="tool">
    <visual>
      <geometry>
        <cylinder length="${T}" radius="${Tr}"/>
      </geometry>
      <origin rpy="${-pi/2} 0 0" xyz="0 ${T/2} 0"/>
      <material name="purple"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="${T}" radius="${Tr}"/>
      </geometry>
      <origin rpy="${-pi/2} 0 0" xyz="0 ${T/2} 0"/>
    </collision>
  </link>

</robot>
```