UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITA' DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Algoritmi e software di controllo per la trazione del rover "Morpheus"

Relatore : Prof. Stefano Debei

Correlatore : Ing. Marco Pertile

Enrico Morellato

mat. 1066927

Anno Accademico 2015/2016

Abstract

In questo progetto di tesi si è andati a studiare un software di controllo per la trazione di Morpheus, un prototipo di rover marziano in fase di progetto presso l'Università di Padova.
Questo rover è equipaggiato con sei ruote fisse in una configurazione skid-steering, e il loro collegamento al corpo del rover è assicurato da un sistema di bilancieri.
L'obiettivo del progetto è stato la scrittura di un codice che, dati degli input da tastiera inseriti dall'operatore, andasse ad individuare le velocità da dover applicare ai gruppi destro e sinistro di ruote. Infatti, come scelta di progetto, si è deciso di utilizzare un modello cinematico semplice, che va a considerare le ruote come suddivise in gruppi destro e sinistro, in una configurazione di guida che si definisce differential drive, ossia che permette il moto applicando velocità differenti ai due gruppi di ruote. Ciò, d'altra parte, costituisce anche il limite di tale algoritmo, in quanto non va a considerare la componente di slittamento, importante per il moto in configurazione skid-steeer.
Tale codice è stato inserito nella scheda elettronica UDOO, della quale si sono studiate varie possibili configurazioni di utilizzo, la quale, tramite dei microcontrollori, andrà a collegarsi ai motori per trasmettere i comandi voluti.

Indice

Introduzione

Morpheus è un progetto condotto da un team di studenti dell'Università di Padova con lo scopo di progettare e costruire un prototipo di rover marziano per poter partecipare all'edizione 2016 dello ERC (European Rover Challenge). Per la realizzazione di tale progetto si è suddiviso il team in vari gruppi ognuno con determinati compiti da svolgere.

Questo progetto di tesi ha come obiettivo principale la scrittura di un codice che permetta, dati degli input da tastiera da parte dell'operatore, di ottenere la movimentazione del rover secondo i comandi che gli sono stati impartiti.

Per ottenere questo si è utilizzata una scheda elettronica, la UDOO Quad, che è stata programmata con un algoritmo che da in uscita le velocità richieste ai gruppi destro e sinistro di ruote. Per ottenere ciò si è anche definito un modello cinematico che, date le velocità lineare e di rotazione del rover, le relaziona con le velocità angolari delle ruote, secondo una configurazione detta differential drive.

La tesi si può dire essere strutturata in vari capitoli, che ora si vanno brevemente a descrivere.

Nel primo capitolo si va a discutere in maniera generale il progetto Morpheus e la competizione a cui parteciperà. Qui sono indicati i vari obiettivi e specifiche che il rover deve possedere.

Successivamente si da una panoramica della meccanica del robot (il progetto meccanico è stato seguito da un altro gruppo di studenti), che poi si concentra nella descrizione della parte inerente al sistema di locomozione.

Nel secondo capitolo si va invece a descrivere il sistema dell'elettronica, prestando particolarre attenzione alla scheda elettronica UDOO Quad, ai microcontrollori e ai motori, i quali vanno a definire la parte che si occupa della trazione del rover.

Qui si è riportata un'accurata analisi della caratteristiche della scheda, ossia il cuore del sistema di controllo per la trazione, andando a riportare le varie configurazioni che si sono testate in ambito di interfaccia tra la scheda stessa ed il pc dell'operatore.

Il terzo capitolo va a trattare la cinematica del rover, parte importante in quanto inserita all'interno del codice che si è scritto.

Questo comincia con una breve introduzione riguardante la robotica mobile per andarsi poi a soffermare sull'analisi cinematica per la definizione di un modello e il suo inserimento all'interno del sistema di controllo. Terminata la parte di introduzione, si è poi andati a definire un modello cinematico per il rover, che viene semplificato in una configurazione differential drive, semplice ma che ben rappresenta comunque il problema.

Il quarto capitolo è quello in cui si vanno a descrivere i codici definiti durante il periodo di tesi, con una breve introduzione relativa a C++, ossia al linguaggio con cui si sono scritti gli algoritmi. Questa parte è suddivisa in due sezioni, che vanno a definire l'evoluzione che il codice ha avuto in base all'aggiornamento della configurazione del rover.

Nella prima parte, si sono andati a scrivere due codici separati che sono stati inseriti nelle due CPU della scheda. Nella seconda parte, invece, il codice scritto è stato uno solo ed è stato inserito nel processore principale della UDOO.

Prima della fine del capitolo si vanno anche ad introdurre la problematiche che si sono riscontrate durante la scrittura del codice all'interno del sistema operativo Ubuntu.

Alla fine di questo, nell'appendice, sono riportati i vari codici definiti durante la fase di programmazione.

Capitolo 1

Il rover e il sistema di trazione

1.1 Il progetto Morpheus

Morpheus, nato dall'omonimo progetto, è il prototipo di un rover marziano ideato da un team di studenti dell'università di Padova con lo scopo di partecipare all'edizione 2016 dello European Rover Challenge (ERC), manifestazione indirizzata agli studenti con lo scopo di progettare e costruire un prototipo di rover marziano, il quale dovrà affrontare e superare determinati requisiti imposti dalla giuria.

I requisiti che tale robot dovrà rispettare sono così suddivisibili:

- generali: dev'essere una piattaforma mobile autonoma, durante le varie operazioni non devono esserci connessioni con sorgenti esterne;
- peso: durante l'esecuzione dei vari task il peso del rover non deve superare i 50 Kg di peso, senza considerare le attrezzature per la manutenzione, le parti di riserva e i vari elementi non montati.

  Se il rover dovesse pesare di più verrà applicata una penalità nel punteggio mentre se dovesse pesare meno del limite questo verrà considerato con dei punti bonus;
- velocità: la velocità del rover dev'essere inferiore ai 3 Km/h;
- controllo e range per le operazioni: il controllo va effettuato via radio in tempo reale; la distanza massima che percorrerà il rover sarà 100 m dal punto di partenza, il quale non disterà più di 50 m dalla control station area, dove sarà disposto l'equipaggiamento per la comunicazione;
- autonomia: include la differenziazione tra vari stati e comandi;
- pulsante d'emergenza: in una zona facilmente accessibile bisogna disporre un interruttore d'emergenza, il cui scopo è quello d'isolare le batterie del sistema;
- comunicazioni: le comunicazioni radio col rover devono usare frequenze legalmente accessibili e livelli di potenza leciti;

- budget: il costo netto per la realizzazione del rover non deve essere superiore a 15000 €.

Quelli sopra elencati sono i vincoli per la realizzazione del rover, inoltre esso dovrà svolgere alcuni compiti durante le varie prove. I vari obiettivi richiesti possono essere così riassunti:

- obiettivo scientifico: geologia.
  Tale obiettivo richiede l'acquisizione di tre diversi campioni di "suolo marziano" , ossia un campione di roccia, uno di suolo ed uno sempre di suolo ma preso ad una certa profondità.
  Per questo sarà necessario che il rover possieda un braccio dotato di uno strumento per poter raccogliere i campioni e di un trapano per poter effettuare la trivellazione del terreno;
- interventi di manutenzione: per quest'obiettivo bisogna dimostrare l'abilità del rover nel poter intervenire su griglie elettriche per poter modificare componenti elettriche.
  Bisognerà usare il sottosistema atto alla manipolazione per intervenire su interruttori e settarli nella configurazione richiesta, oltre che misurare parametri elettrici;
- assistenza: qui bisogna effettuare un'operazione di consegna di un componente in un determinato luogo, posizionarlo con un'adeguata orientazione e ritornare al punto di partenza consegnando un report di quanto svolto;
- navigazione: in quest'obiettivo bisogna dimostrare la capacità del rover di raggiungere una destinazione senza o con limitata supervisione. Per ottenere questo si dovrà disporre di un sistema che permetta all'operatore di navigare senza l'accesso a dati visivi (come lo streaming video, foto, ecc..); fondalmentalmente bisognerà che i dati vengano processati a bordo dei veicolo e che inviino all'operatore informazioni riguardo la localizzazione e il supporto per l'operazione.

1.2 Il rover Morpheus e la meccanica

Per rispettare i vari vincoli imposti e per raggiungere i vari obiettivi si è arrivati a progettare il rover Morpheus, un prototipo simile a quelli per l'esplorazione planetaria, il quale sarà in grado di svolgere compiti simili a quelli che supporteranno le future missioni umane a Marte.

Tale rover sarà comandato in remoto da un operatore e sarà in grado di raccogliere campioni di suolo e di roccia, di effettuare trivellazioni, di guidare in maniera autonoma su terreni duri e di svolgere operazioni di assistenza e manutenzione.

Importante nell'ambito della competizione è la presenza di un braccio robotico, dotato di tre differenti manipolatori intercambiabili, in base all'obiettivo da dover soddisfare, e di un trapano, necessario alla trivellazione del terreno per poter ottenere campioni del suolo ad una data profondità.

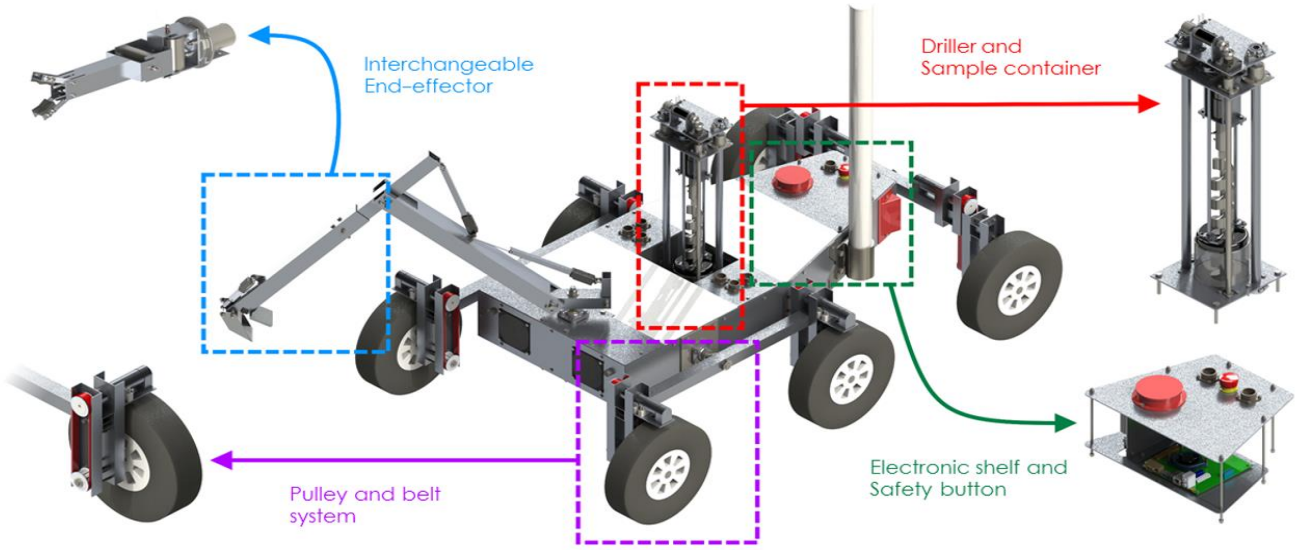

Fig. 1.1 Immagine del rover con inclusi i sottosistemi più importanti. Fonte: Morpheus Team 2015

Il telaio è stato realizzato con profilati di acciao (di dimensioni 20x20x2 mm), i quali contribuiscono a contenere il peso, che è di circa 46 Kg. Il telaio è ricoperto da dei pannelli sempre realizzati in acciao.

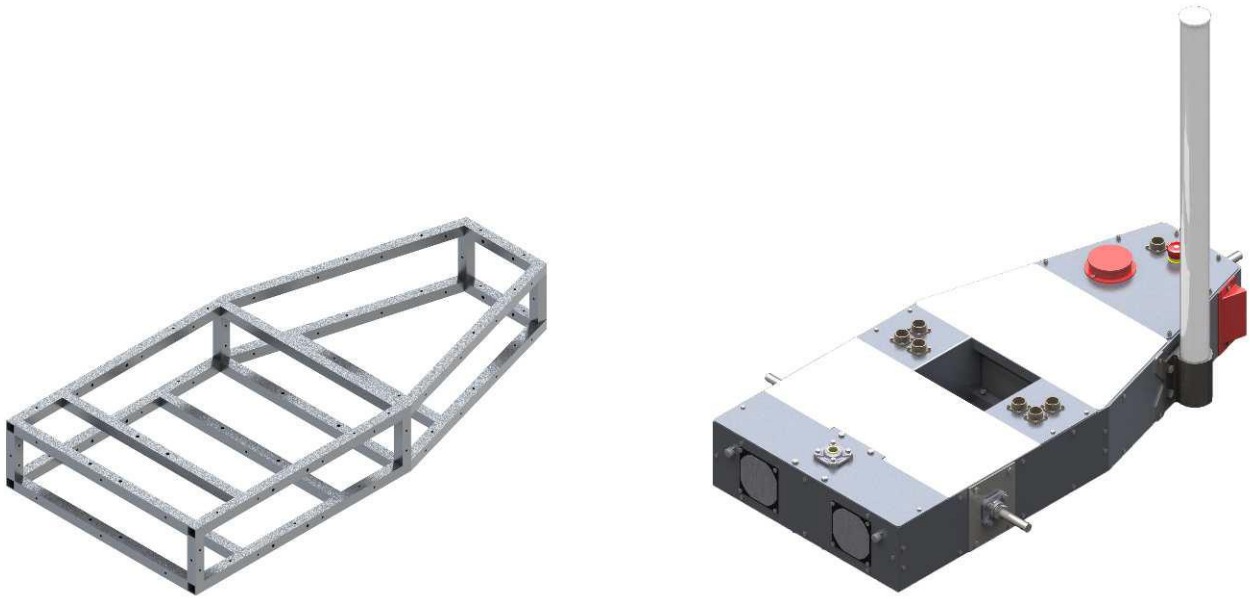

Fig. 1.2 Immagine del telaio del rover (si nota la struttura formata da profilati d'acciaio) a sinistra, mentre a destra si ha una visione del telaio con i pannelli di copertura installati. Fonte: Morpheus Team 2015

Il telaio ha il compito di sopportare tutti i carichi strutturali ed è suddiviso in due parti principali: quella frontale di forma rettangolare, mentre quella posteriore ha la forma di un trapezio isoscele (con angoli alla base di 16°).

I vari pannelli di copertura permettono anche il montaggio dei connettori, delle ventole e delle PCB (Printed Circuit Board, ossia i circuiti stampati) e inoltre di isolare l'interno della struttura dall'ambiente esterno. Sul rover verrà inoltre installato il braccio robotico, il quale svolgerà alcuni dei compiti richiesti dalla ERC, tra i quali il requisito di manutenzione ed assistenza e quello scientifico.

Tale braccio sarà rimovibile, così come la sua terminazione (end-effector); infatti, sono state progettate due diverse estremità, una a 3 gradi di libertà per l'obiettivo scientifico, l'altra a sei per svolgere le operazioni di manutenzione ed assistenza.

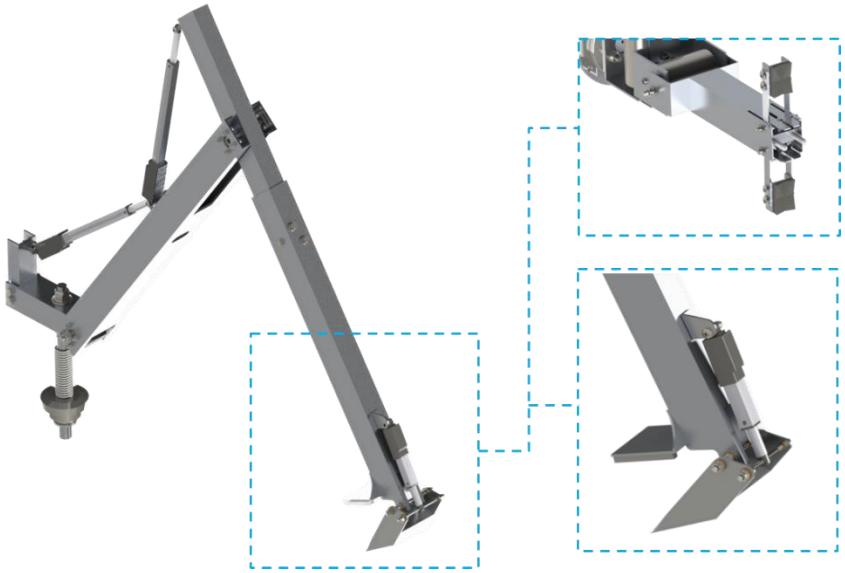

Fig. 1.3 Immagine del braccio e dei due end-effectors che verranno utilizzati. Fonte: Morpheus Team 2015

Inoltre sarà montato anche un trapano, il quale sarà necessario per la trivellazione del terreno per poter estrarre campioni del sottosuolo.

Fig. 1.4 Immagine del trapano che verrà usato per il task di trivellazione. Fonte: Morpheus Team 2015

Per quanto riguarda la locomozione, il rover si muove grazie a sei ruote di diametro 26 cm, disposte in configurazione skid-steer, per cui si hanno ruote non sterzanti che permettono la rotazione grazie alla differente velocità delle due singole ruote. Le ruote sono connesse, a gruppi di due, a dei bilancieri, i quali le collegano al telaio tramite un perno supportato da due cuscinetti. Inoltre su ogni ruota c'è un meccanismo di trasmissione a puleggia per il sistema di connessione, grazie al quale non si va a sollecitare troppo i motori.

L'utilizzo di questa configurazione di ruote e bilancieri ha reso non necessaria la progettazione delle sospensioni, in quanto è possibile comunque poter affrontare terreni scoscesi garantendo un'adeguata aderenza al suolo.

Infatti, il rover è progettato per raggiungere la massima velocità di 1 m/s (3.6 Km/h) nel tempo di 5 secondi, su un suolo con pendenza massima pari a 15° e con un angolo di roll massimo pari a 50°.

Con tale design è possibile effettuare una pura rotazione attorno al centro.

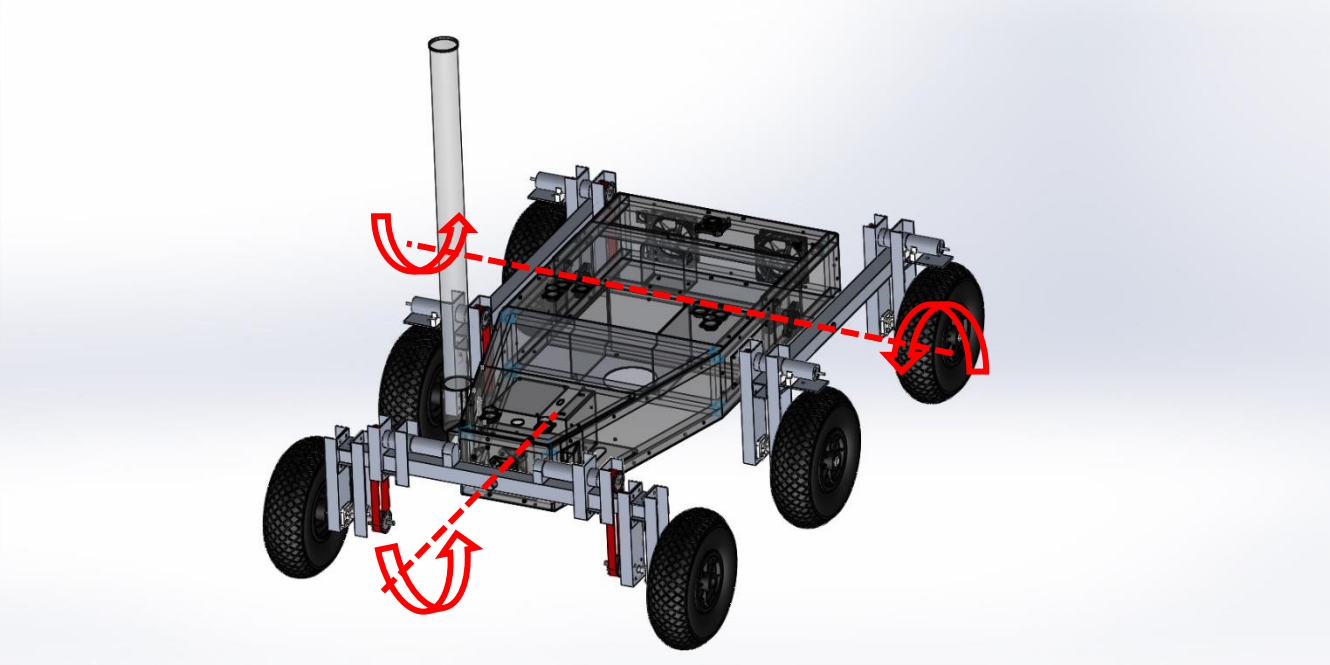

Fig. 1.5 Immagine del rover con il sistema di bilancieri per la locomozione e la rappresentazione dei vari gradi di libertà. Fonte: Morpheus Team 2015

Passando ad analizzare i bilancieri, avremo che due di questi sono posizionati lungo il fianco del rover, mentre il terzo è posizionato nella sezione posteriore in maniera tale da poter compensare

l’angolo di roll (causato dalla presenza di terreni accidentali, con cunette e dossi) e permettere che le sei ruote siano sempre a contatto col terreno.
I tre bilancieri ruotano attorno assi diversi e hanno delle configurazioni leggermente differenti.
Analizzando il bilanciere laterale, esso è costituito da due barre verticali connesse ad una orizzontale mediante collegamenti con bulloni; nella parte superiore delle barre verticali sono collocati i motoriduttori. La puleggia è fissata meccanicamente all’albero del riduttore tramite una chiave.
Delle cinghie permettono la trasmissione del moto alla puleggia inferiore, la quale è fissata al mozzo della ruota mediante un grano di fissaggio. Anche il giunto tra mozzo e ruota consiste in una colla per accoppiamento cilindrico e permette di prevenire lo slittamento relativo tra i due, aumentando l’attrito.
Il supporto orizzontale è costituito da un tubolare a sezione quadrata di dimensioni 30x30x2 mm e lungo 600 mm, mentre le due barre verticali sono formate da un tubolare con sezione a C di dimensioni 50x30x2 mm e aventi lunghezza 210 mm.

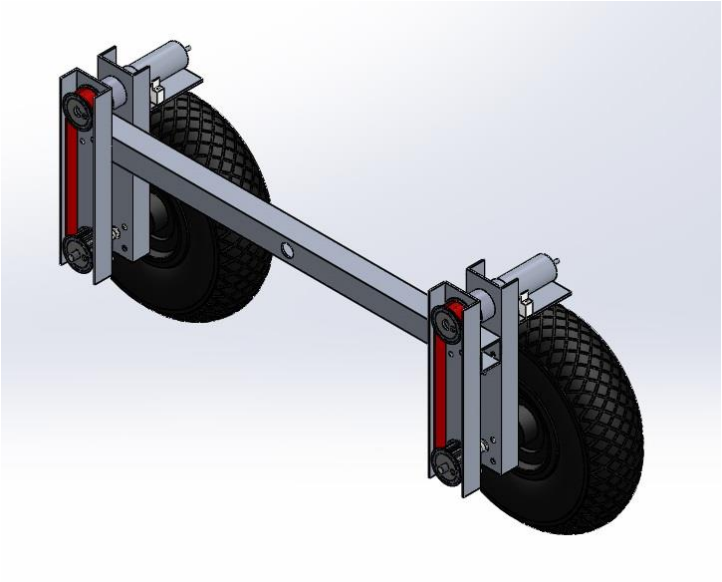

Fig.6 Il bilanciere laterale. Fonte: Morpheus Team 2015

Analizzando il bilanciere posteriore, esso a livello costruttivo e di componenti è molto simile a quello laterale, eccezion fatta per la posizione dei motori.

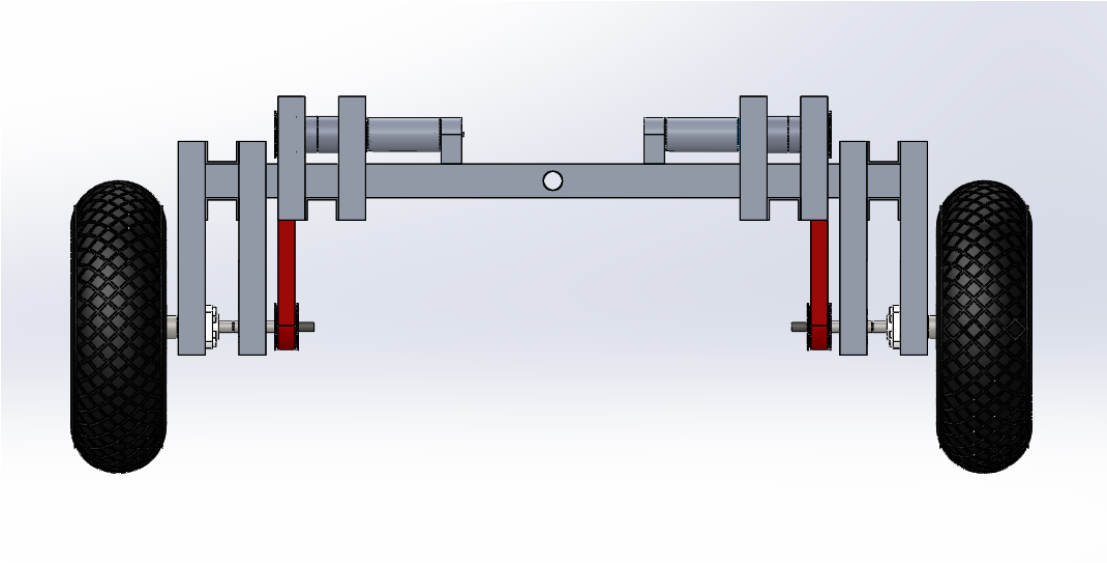

Fig. 1.7 Immagine del bilanciere posteriore. Fonte: Morpheus Team 2015

Capitolo 2

Elettronica del rover Morpheus

2.1 Panoramica sul sistema dell'elettronica

Le due schede usate per il controllo del rover Morpheus sono la JETSON TK1 e la UDOO.

La prima è una scheda della NVIDIA ed è quella principale del rover; controlla i sensori, processa la immagini per il sistema di stereovisione e gestisce la comunicazioni tra Morpheus e la base station.

La seconda controlla il braccio robotico e i motori per la locomozione.

Il sistema di telecomunicazione è gestito da due antenne omnidirezionali, una per il robot e una per la stazione di controllo, che permettono la comunicazione nel raggio di 1 Km.

Il sistema di potenza del rover consiste in batterie commerciali ai polimeri di ioni di Litio e dovrebbe assicurare un tempo di lavoro di circa un'ora.

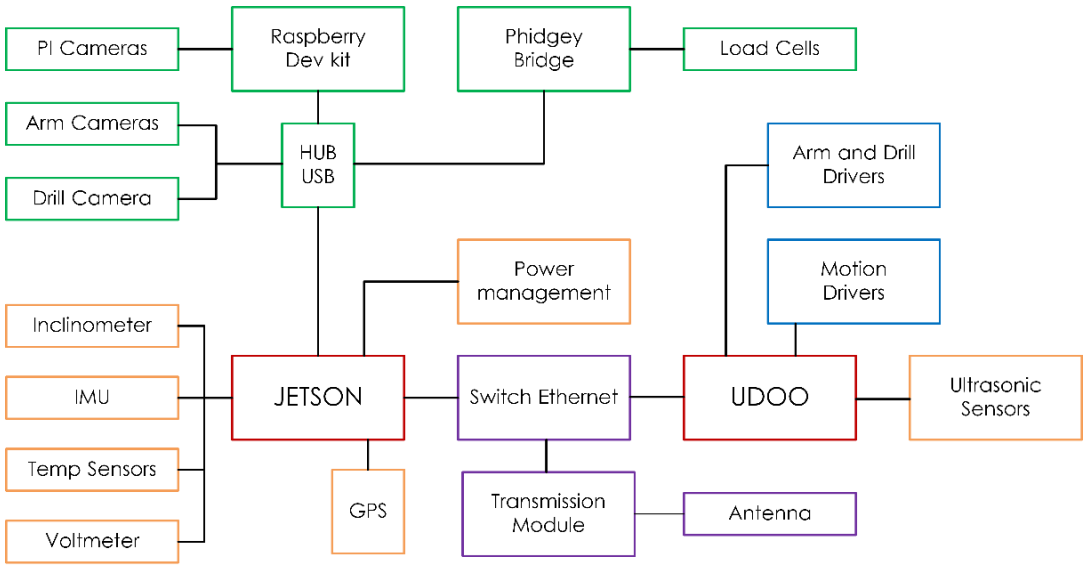

Fig. 2.1 Schema funzionale del rover, con i vari sensori, driver ed il sottosistema di comunicazione.
Fonte: Morpheus Team 2015

2.1.1 La scheda Jetson

La NVIDIA JETSON TK1 è il cuore del rover Morpheus e sfrutta una CPU 4-PLUS-1 Cortex A15 “r3”.

Fig. 2.2 NVIDIA JETSON TK1.

Questa è connessa a diversi sensori, i quali sono usati per la gestione interna del rover (controllo temperature), localizzazione (gestisce i segnali GPS, IMU e stereoimmagini) e la movimentazione del rover.

Tra i vari dispositivi collegati alla Jetson sono da citare:

- tre webcam, due per il controllo del braccio mentre la terza viene usata per il trapano;
- un sistema di stereovisione, necessario per il controllo della navigazione autonoma, composto da una scheda Raspberry-PI (un calcolatore implementato in una singola scheda elettronica) e due fotocamere;
- due celle di carico (con capacità 5 Kg ±0.05 %) connesse alla scheda con un Phidget-Bridge.

Tutte le informazioni raccolte dai sensori vengono inviate alla base station in tempo reale tramite un’antenna omnidirezionale AirMax Omni AMO-5G10.

Un altro dei compiti rilevanti di questa scheda è relativo alla gestione ed al controllo della potenza; infatti comunica con essa mediante una scheda dedicata, composta da vari convertitori DC/DC che inviano i voltaggi necessari al rover e ai suoi vari sottosistemi.

I voltaggi e i consumi di potenza vengono costantemente monitorati per prevenire un eccessivo consumo di energia da parte delle batterie; in questo modo questi possono essere controllati e corretti in caso di funzionamenti anomali o irregolari.

Inoltre tra le batterie e il blocco di distribuzione della potenza sono presenti filtri (necessari per eliminare rumori alle alte frequenze e correnti inverse) e un sistema d'emergenza, il quale in caso di bisogno interrompe l'erogazione di potenza.

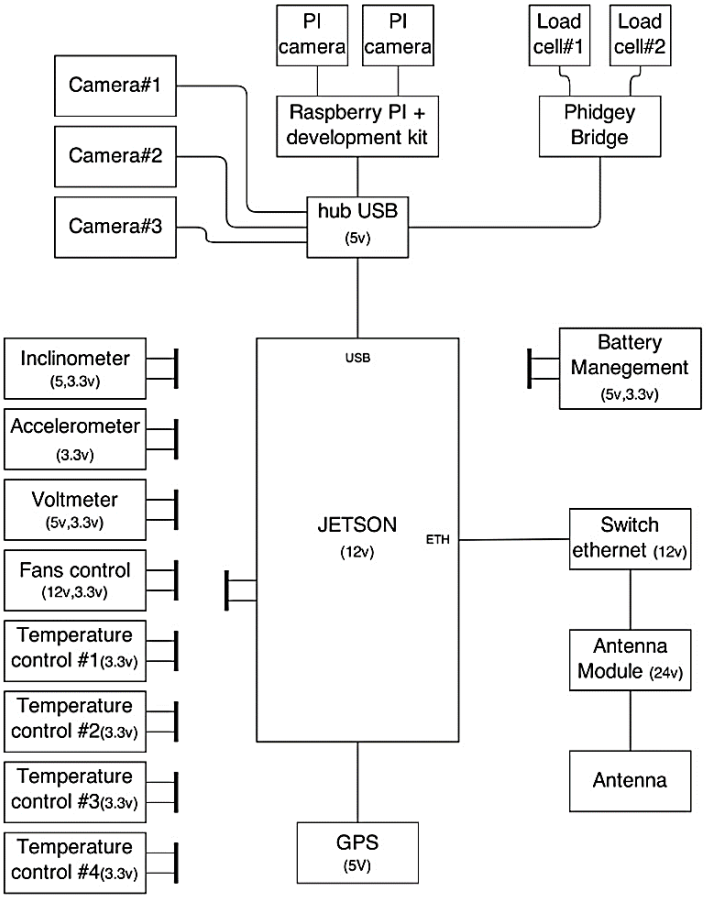

Fig. 2.3 Architettura della scheda principale Jetson con le connessioni ai vari sottosistemi e sensori.
Fonte: Morpheus Team 2015

2.2 La scheda UDOO, i microcontrollori e i motori

La locomozione del rover e la movimentazione del braccio robotico sono controllate dalla scheda elettronica UDOO.

Per quanto riguarda la locomozione, la scheda, dati degli input (da tastiera o da joystick), andrà a produrre delle uscite che poi i microcontrollori ed i driver dei motori andranno a convertire così da produrre la movimentazione delle ruote.

2.2.1 UDOO

La scheda UDOO è il cuore del sistema di locomozione del rover Morpheus, in quanto include i vari algoritmi per il controllo della trazione.

La scelta tra i vari tipi disponibili è ricaduta sulla UDOO Quad, la quale è equipaggiata con due CPU: infatti integra un single board computer (un processore ARM Freescale Cortex-A9 i.MX6 Quad Core) e un microcontrollore compatibile con Arduino 2 con un ARM dedicato Atmel SAM3X8E CPU.

Questa scheda è un'ottima scelta per sviluppare software e per design.

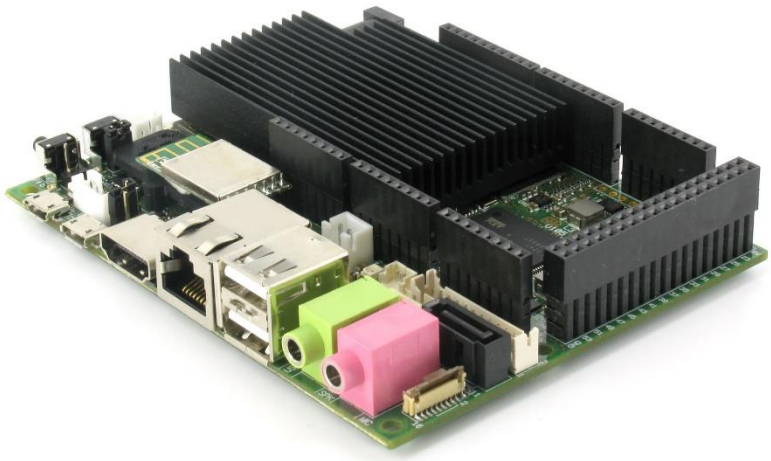

Fig.2.4 La scheda UDOO.

Si va ora ad analizzare la varie parti che compongono la scheda:

- il processore ARM Freescale Cortex-A9 i.MX6 Quad Core;
- la CPU Atmel SAM3X8E ARM Cortex-M3 (lo stesso di Arduino 2);
- RAM DDR3 1GB;
- 76 GPIO disponibili con i pinout compatibili con Arduino;
- porta HDMI;
- 2 porte micro USB;
- 2 USB 2.0 tipo A;
- due jack per Analog Audio e Mic;
- lettore per schede microSD (la scheda SD è necessaria in quanto è li che si va ad installare il sistema operativo);
- presa Ethernet RJ45;
- modulo WiFi;
- connessione SATA;
- l'attacco per l'alimentazione elettrica.

Parlando delle due CPU, il processore i.MX6 è il cuore del sistema operativo della UDOO, infatti gestisce:

- il sistema operativo (possono esserne installati diversi tipi, tra i quali Ubuntu e Android);
- le operazioni audio/video (video HDMI, LVDS,input e output audio);
- le porte USB;
- la connessione SATA.

Il processore SAM3X, invece, è compatibile con Arduino Due ed è il cuore del supporto agli input ed output della scheda. Questo può essere usato da solo come una scheda Arduino con la possibilità di connettere qualunque dispositivo compatibile con esso.

Il modo migliore per sfruttare i due processori e per ottenere progetti avanzati è, però, quello di metterli in comunicazione: infatti la UDOO comprende un canale seriale per interconnettere i due.

Infatti entrambe le CPU possono ricevere ed inviare dati attraverso la linea seriale, l'unica cosa a cui bisogna prestare attenzione è che i due comunichino con la stesso baudrate.

In generale è possibile quindi avere tre scenari possibili (si considera Linux come sistema operativo installato):

- comunicazione unidirezionale da Linux ad Arduino: si considera nella parte Arduino l'accensione di un LED quando uno script viene lanciato nel sistema operativo (installato nella parte iMX6).

  Bisogna settare adeguatamente la porta seriale in Linux, selezionandola e indicando il baudrate scelto. Poi si va a lanciare lo script nella parte i.MX6, il quale andrà a scrivere i dati sulla porta seriale. Successivamente si passa all'interno della parte Sam3x e si carica lo sketch Arduino, questo si metterà in ascolto nella porta seriale aspettando l'informazione inviata da iMX6.
- comunicazione unidirezionale da Arduino a Linux: questo è il caso, ad esempio, in cui un'applicazione in Linux darà come output dei dati raccolti da dei sensori connessi alla parte Arduino. Anche in questo caso è importante impostare la porta seriale al baudrate settato dallo sketch. Il Sam3x raccoglierà dei dati dai propri sensori e li invierà, propriamente codificati, via seriale. Dal lato Linux si può impostare un server locale che resti in ascolto per i dati seriali e li vada a mettere in un database locale.
- Comunicazione bidirezionale: in questa situazione i due sistemi leggono e ascoltano reciprocamente. Si considera, ad esempio, un'applicazione integrata: un sensore che legge dati ambientali come temperatura e pressione. Si carica lo sketch di Arduino, il quale invierà entrambe le letture dei sensori via porta seriale ogni secondo. Nella parte i.MX6 si va a settare un server che immagazzina i vari dati letti in un database. Successivamente si vuole avere come output i valori medi di temperatura e pressione con una striscia LED connessa alla parte Arduino.

  Si va a configurare il server per scrivere i dati che verranno visualizzati sulla striscia LED allo stesso baudrate e si integra il precedente sketch di Arduino così da aggiungere l'ascolto su seriale e la scrittura sulla striscia LED. Si vede che si usa la connessione seriale in entrambe le direzioni e anche gli script Linux e Arduino possono ricevere ad inviare dati allo stesso momento.

  Inoltre la UDOO QUAD è in grado di comandare i propri GPIO in modo diretto (come altre schede integrate); questo è il metodo più semplice per controllarle però permette di farlo solo

mediante il sistema Linux installato sull' i.MX6. Con questo diventa possibile controllare i GPIO mediante i.MX6 mentre essi sono pilotati anche da un controller compatibile con Arduino SAM3x.

Bisogna però prestare attenzione ad una cosa: settare lo stesso pin su diversi stati nello stesso momento, oppure controllarlo con entrambe le CPU porterà al danneggiamento della scheda; per questo bisogna assicurarsi che il pin/GPIO possa avere un solo stato nello stesso tempo e che sia controllato da un solo processore alla volta.

Il sistema operativo montato sulla scheda che si è deciso di utilizzare è UDOObuntu, di base Linux.

Questo gira sfruttando il processore i.MX6 ed è stato installato su una scheda microSD da 8 GB.

Una volta installato il sistema operativo si è proceduto con alcune operazioni preliminari prima di iniziare il lavoro vero e proprio.

La prima di queste è stata la configurazione del Minicom, avvenuta collegando la scheda al PC usando un cavo micro USB. Con quest'operazione si è poi modificato il setup della porta seriale.

Il potersi collegare all'interfaccia seriale da la possibiltà, inoltre, di potersi connettere alla scheda via SSH senza disporre di una rete, di poter programmare il processore Sam3x e di poter accedere alla console di debug per risolvere dei problemi che possono manifestarsi.

Oltre a questo tipo di connessione col pc (ossia mediante cavo seriale) è possibile poter collegare la UDOO tramite SSH.

Per poter sfruttare questo tipo di connessione bisogna conoscere l'indirizzo IP della scheda, cosa ottenibile in diversi modi.

SSH è un protocollo di comunicazione via rete che permette di poter controllare la shell in remoto, ossia senza dover essere fisicamente collegati alla scheda.

Per poterlo fare bisogna però soddisfare alcune "richieste": il pc e la UDOO devono esser connesse allo stesso network, non ci devono essere firewall attivi che possano bloccare la comunicazione tra i due dispositivi.

Una volta verificati questi, si accede al terminale del pc e si inserisce la seguente sintassi:

ssh username@Udoo_Ip

Successivamente si va ad inserire la password; si sono mantenute le impostazioni di default, per cui l'username è ubuntu e la password è ubuntu.

La configurazione utilizzata per la scheda elettronica durante la fase di programmazione e le prove consiste in:

- scheda microSD con installato il sistema operativo;
- tastiera e mouse collegati tramite le porte USB;
- connessione ad uno schermo mediante cavo HDMI; si è considerato il fatto di usare una connessione che permettesse di non usare uno schermo aggiuntivo (visto che sarebbe inutile in quanto installato sul rover) ma di sfruttare quello del pc.

  Nel caso usato in laboratorio si è sfruttata la rete internet disponibile per ottenere una connessione VPN ed usare così lo schermo del computer.

  Nel caso reale, invece della connessione internet, si potrà sfruttare l'antenna installata sul rover per generale un collegamento tra UDOO e il pc dell'operatore.

2.2.2 I microcontrollori

I microcontrollori (sono stati ordinati ma non sono ancora disponibili) scelti per collegare la scheda elettronica ai driver dei motori sono degli Atmel Atxmega256a3u.

Questi sono dei dispositivi elettronici integrati su singolo chip, evoluzione alternativa al microprocessore e usato generalmente in sistemi embedded (ossia applicazioni specifiche di controllo digitale). È progettato per interagire tramite un programma residente nella propria memoria interna e mediante l'uso di pin specializzati o configurabili da un programmatore.

Possiedono un'ampia gamma di funzioni di controllo e comando, sia analogiche che digitali, integrate nello stesso chip, e ciò permette l'uso dei MCU in sostituzione di schede elettroniche cablate (più complesse e costose).

Nonostante sia un'alternativa al microprocessore, si differenzia da esso per alcune caratteristiche: il microprocessore, per quanto potente, integra sul chip solo la logica di elaborazione, mentre richiede sempre delle unità esterne (memorie, gestori di segnali, dispositivi periferici per poter scambiare informazioni e poter interagire con l'esterno); il microcontrollore, invece, è un sistema completo, che integra in uno stesso chip il processore, la memoria permanente, la memoria volatile e i canali (pin) di I/O, oltre ad eventuali altri blocchi specializzati.

I processori classici, poi, sono adatti ad un uso generale (general purpose), mentre i microcontrollori sono progettati per ottenere la massima autosufficienza funzionale ed ottimizzare il rapporto prezzo/prestazioni in uno specifico campo d'applicazioni.

Anche l'esecuzione di programmi applicativi si appoggia su un'architettura hardware differente rispetto ai microprocessori, in quanto, utilizzano come memoria un dispositivo di memoria ROM ed è quindi in realtà un firmware.

Analizzando l'architettura di un microcontrollore, essa prevede un insieme di moduli fissi (comuni a tutti i modelli) ed una serie di possibili estensioni in funzione del costruttore, del prezzo e della fascia applicativa:

- unità elaborativa: CPU;
- memoria di programma: ROM, FLASH;
- memoria dati: RAM;
- oscillatore interno od esterno;
- porte di I/O e/o GPIO configurabili;
- gestione interrupt;
- moduli aggiuntivi.

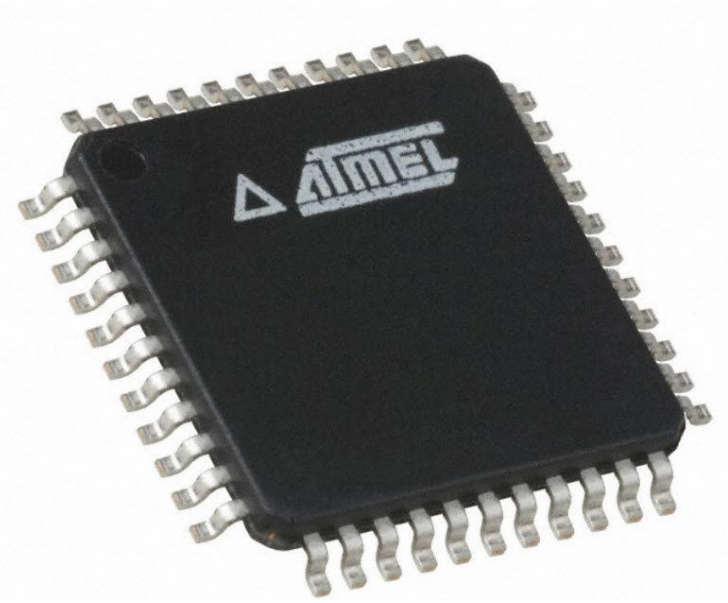

Fig. 2.5 Microcontrollore.

Si è deciso di usare il microcontrollore della Atmel per il suo basso consumo di potenza e per le sue elevate performance. In ingresso riceverà gli output della scheda elettronica e li convertirà in delle uscite leggibili dai driver dei motori; il collegamento tra UDOO e il MCU sarà effettuato tramite cavo USB.

2.2.3 I motori

I motori che si è scelto di utilizzare per azionare le ruote del rover sono dei motori brushless della Maxon, nello specifico il modello EC-max 30 Φ30 da 60W.
Però l'azionamento vero e proprio non è composto esclusivamente dai motori, ma anche da riduttori e da controller digitali (uno per ogni ruota).
Questi pezzi purtroppo sono stati solo ordinati e non sono disponibili in laboratorio.

Si comincia analizzando il riduttore (sempre della Maxon), planetario modello GP 32 HP Φ32 mm, in metallo, nella variante High Power.
Prima di analizzare le caratteristiche di questo, si da una breve descrizione dei riduttori planetari.
Il riduttore epicicloidale (o planetario) è un organo meccanico che attraverso un semplice meccanismo riesce a modificare i rapporti di velocità tra l'albero d'ingresso e quello di uscita. I modelli geometricamente coassiali, come quello adottato in questo caso, sono costituiti da un ingranaggio solare, un portasatelliti ed una corona a dentatura interna; questi elementi trasmettono il moto grazie ad ingranaggi satelliti, rotanti con interasse fisso, solidale con il portasatelliti.
Tipicamente l'ingresso del moto avviene dall'ingranaggio solare ed è trasmesso attraverso i satelliti al portasatelliti, solidale con l'albero di uscita. Il numero di denti degli ingranaggi e della corona a dentatura interna determina il rapporto di riduzione.
I vantaggi nell'utilizzo di questo tipo di riduttore sono direttamente derivati dal suo modo di funzionare e consistono nella possibilità di avere:

- elevati rapporti di riduzione; questo in termini pratici consiste nell'ottenere rispetto ad una riduzione ad ingranaggi, a parità di rapporto, prestazioni più elevate e in molti casi l'utilizzo di minori stadi di riduzione;

- elevate coppie da poter trasmettere; ciò porta a dimensioni e pesi più contenuti, in quanto si hanno ingranamenti multipli che riducono quindi le forze agenti sui denti;
- elevati carichi da sopportare sull'albero in uscita.

Andando a considerare il riduttore scelto, i dati più significativi sono:

- rapporto di riduzione 86:1;
- composto da 3 stadi;
- peso di 210 g.

Passando a descrivere i motori, è stato scelto il modello EC-max 30 Φ30 da 60W, con inclusi sensori ad effetto Hall. Tale modello fa parte della categoria dei motori brushless, ossia un motore elettrico a corrente continua con il rotore a magneti permanenti (e quindi sprovvisto di avvolgimento) e lo statore a campo magnetico rotante. A differenza di un motore a spazzole non ha quindi bisogno di contatti elettrici striscianti (spazzole) sull'albero motore per funzionare. La commutazione della corrente circolante negli avvolgimenti dello statore, e quindi la variazione dell'orientamento del campo magnetico da essi generato, avviene elettronicamente. Ciò comporta una minore resistenza meccanica, elimina la possibilità che si formino scintille al crescere della velocità di rotazione, e riduce notevolmente la necessità di manutenzione periodica.
Poiché il motore funziona in corrente continua, per realizzare la rotazione del campo magnetico generato nello statore, un circuito elettronico, composto da un banco di transistor di potenza comandati da un microcontrollore che controlla la commutazione della corrente, comanda l'inversione di corrente e quindi la rotazione del campo magnetico. Dato che il controllore deve conoscere la posizione del rotore rispetto allo statore per poter determinare l'orientamento da dare al campo magnetico, esso viene solitamente collegato a un sensore ad effetto Hall.
Il primo grosso vantaggio riguarda la vita attesa del motore, dato che le spazzole sono il "punto debole" di un motore elettrico. L'assenza di spazzole elimina anche la principale fonte di rumore elettromagnetico presente negli altri motori elettrici.
L'ingombro è limitato rispetto alla potenza che possono erogare e soprattutto rispetto alla coppia che questi motori riescono ad esprimere. In termini di efficienza, i motori brushless lavorano sempre in condizioni di rendimento ottimali. Non dovendo generare il campo magnetico rotorico

hanno efficienze maggiori. Il principale svantaggio di questo tipo di motori sta nel maggiore costo. A differenza dei motori a spazzole, infatti, il controllo viene effettuato elettronicamente da un controller, un dispositivo elettronico fornito dal costruttore del motore o da terze parti, quindi al costo del motore va aggiunto il costo del sistema di controllo.

Il motore della Maxon viene fornito con collegato un sensore ad effetto Hall; questi sono sensori che sfruttano la seconda legge elementare di Laplace. Una volta alimentati, se sottoposti anche per brevi istanti ad un campo di induzione magnetica, danno in uscita una tensione proporzionale all'intensità del campo. L'elettronica di pilotaggio determina la posizione del rotore a partire dai sensori ad effetto Hall. Da questa, ne deduce l'orientamento da dare al campo magnetico dello statore. Durante la rotazione, l'elettronica di pilotaggio comanda le tre bobine per variare nel tempo l'orientamento del campo in relazione alla posizione del rotore, in modo da trascinarlo nel senso scelto dall'utilizzatore.
Modulando la corrente nelle bobine, l'elettronica può accellerare o rallentare il motore e regolare in questo modo la sua velocità. Può inoltre orientare il campo magnetico in modo da frenare il rotore nel suo movimento fino all'arresto. Limitando la corrente nelle bobine, l'elettronica può egualmente limitare la coppia del motore.

Ora si procede con l'elencare alcuni dati caratteristici del motore da noi utilizzato:

Valori con tensione nominale	
Tensione nominale	24 V
Velocità a vuoto	9340 rpm
Corrente a vuoto	231 mA
Velocità nominale	8040 rpm
Corrente nominale	2.66 A
Efficienza massima	81 %
Dati meccanici	
Tipo di cuscinetto	Cuscinetto a sfera
Velocità max permessa	15000 rpm
Altri dati	
Numeri di paia poli	1
Numero di fasi	3

Tab.1 specifiche più rilevanti per il motore Maxon EC-max 30 brushless.

Fig. 2.6 Motore Maxon EC-max 30.

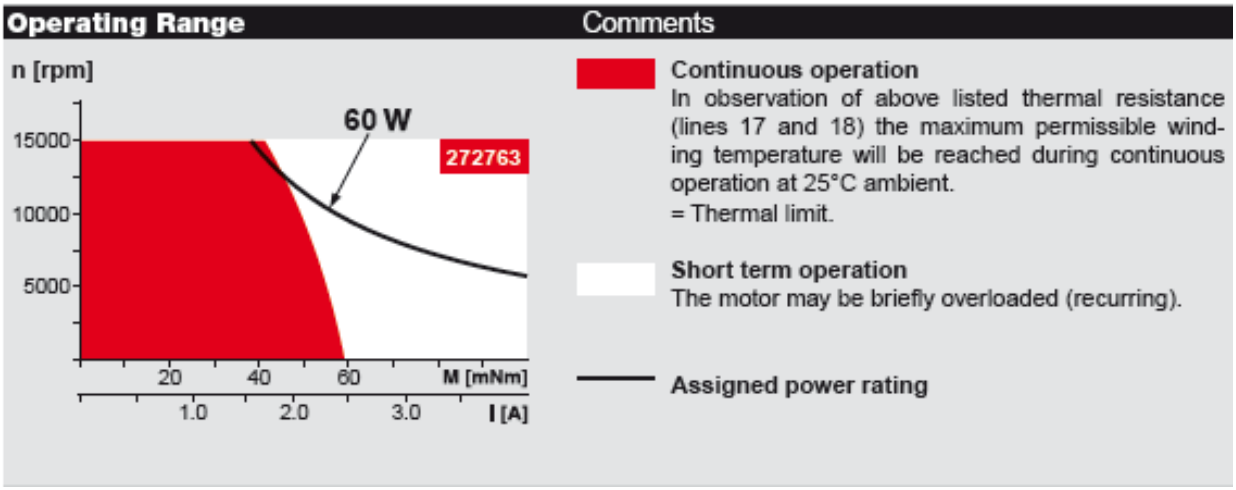

Fig. 2.7 Range operativo per il motore.

Infine si va a dare una descrizione dei controller digitali connessi ai motori, il modello scelto è il DEC (Digital EC controller) module 50/5.

Questo è un controller digitale ad un quadrante per il controllo di motori DC brushless fino a 250 W.

Per poter essere usato bisogna che il motore sia equipaggiato con sensori ad effetto Hall.

La sue caratteristiche principali sono le seguenti:

- il controllo digitale di velocità può funzionare come controllo “closed loop” oppure come “open loop”;
- massima velocità raggiungibile pari a 80000 rpm (con motori ad un paia di poli);
- il valore di input può esser settato attraverso il voltaggio analogico esterno;
- possono esser selezionati tre differenti range di velocità;
- direzione della rotazione può esser preimpostata da un segnale digitale;
- lo stadio d’uscita può esser o meno abilitato;
- la massima corrente limite in output può arrivare fino a 10 A;
- la velocità del motore può esser monitorata con l’output “monitor n”;
- protezioni contro sottovoltaggio, sovravoltaggio e surriscaldamento.

Gli ingressi digitali hanno tre tipi di funzionalità: abilitazione, senso di rotazione, campo di velocità.

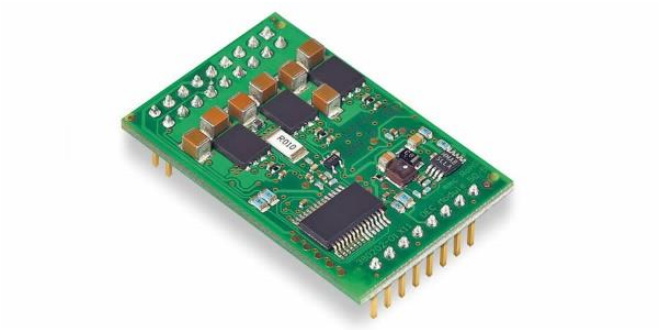

Fig. 2.8 Controller digitale.

CAPITOLO 3

Il modello cinematico del rover

3.1 Introduzione

Per le prime simulazioni e per i primi test (quando saranno disponibili i microcontrollori ed i motori) si è deciso di utilizzare un modello cinematico semplificato che permette di ottenere, dati i comandi di input, come uscite le velocità angolari dei gruppi di ruote destre e sinistre.

Si è deciso di procedere così per il fatto che la configurazione iniziale non prevedeva l'uso dei microcontrollori, per cui era possibile disporre di due sole uscite analogiche (invece che digitali come ora) che hanno portato a dividere le sei ruote in due gruppi, quello delle ruote destre e quello delle ruote sinistre.

Questa è una semplificazione del problema completo, in quanto non va a considerare le asperità del terreno ed il fatto che i bilancieri del rover, muovendosi, fanno cambiare configurazione alle ruote che di conseguenza potranno avere velocità differenti.

Uno dei passi successivi sarà, una volta disponibile il modello cinematico completo, quello di poter ottenere sei uscite digitali che, tramite cavo USB, verranno inviate ai microcontrollori che, insieme ai driver dei motori, faranno si che la velocità desiderata in ingresso si converta in un'adeguata velocità di rotazione delle varie ruote.

Il motivo per cui si è utilizzato un modello cinematico è che i motori sono controllati in velocità.

Prima di analizzare il caso trattato, si va a fare prima una breve introduzione generale della robotica mobile e della cinematica dei rover.

3.1.1 Generalità sulla robotica mobile e sulla cinematica del rover

Parlando dei robot, per molte applicazioni la soluzione migliore per la locomozione (come nel caso di Morpheus) è l'utilizzo di ruote, per le quali esistono diverse configurazioni:

- ruota fissa;
- ruota centrata orientabile;
- ruota scentrata ed orientabile (ruota Castor);
- ruota svedese, avante proprietà omnidirezionali, in quanto si basa sull'idea di poter sfruttare l'attrito anche trasversalmente rispetto alla direzione di moto della ruota.

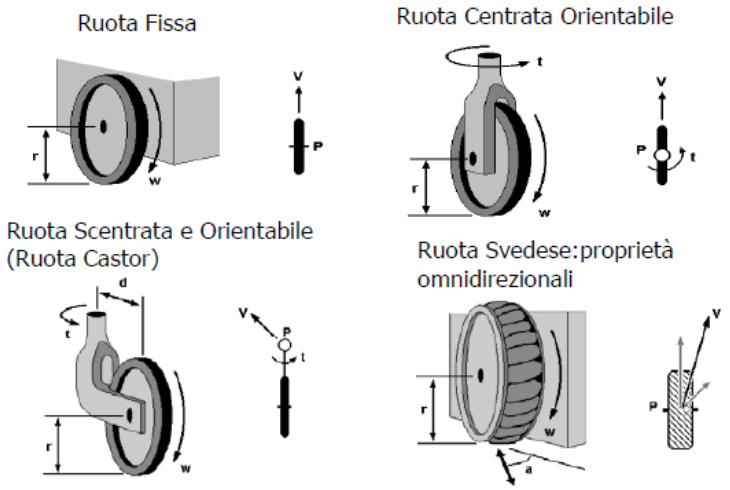

Fig. 3.1 Immagine delle varie configurazioni di ruote.

In base a molteplici fattori è possibile scegliere una configurazione per le ruote.

Una volta nota questa, si va ad introdurre lo spazio delle configurazioni, il quale è caratterizzato dal numero di parametri che servono a localizzare la configurazione del robot ed, ovviamente, dipende dalla sua struttura.

Successivamente, andiamo a descrivere i vincoli, anch'essi essenziali per il modello cinematico.

Si dice vincolo una qualunque condizione imposta ad un sistema materiale che impedisce di assumere una generica posizione e/o atto di moto.

Un sistema materiale si dice soggetto a vincoli olonomi se tra le coordinate del sistema esistono dei legami espressi da relazioni finite (vincoli di posizione) oppure se tra le coordinate del sistema esistono dei legami espressi da relazioni differenziabili integrali finite.

Si dice anolonomo un vincolo in cui la relazione differenziale tra le coordinate non è riducibile a forma finita.

Si vanno ora ad analizzare i vincoli anolonomi applicati alle ruote.
L'ipotesi semplificativa che poniamo è che ogni ruota rotola senza strisciamento, ossia senza slittare ne lateralmente ne longitudinalmente.

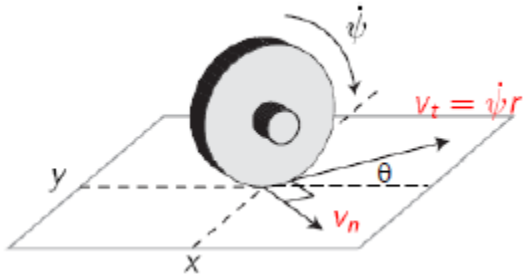

Fig. 3.2 Immagine di una ruota con rappresentati i vincoli.

Ogni ruota introduce un vincolo anolonomo nel sistema in quanto non consente una traslazione normale alla direzione di rotolamento; questo porta ad una limitazione istantanea della mobilità del robot, senza però generalmente precludere la possibilità di raggiungere configurazioni arbitrarie.
Questo fatto assicura il contatto continuo ed è utile per il dead-reckoning (odometria).
Andando a considerare la coordinata x lungo la direzione di t e la y lungo la direzione normale si ottiene:

$$\begin{cases} \dot{x} = v_t \cos\theta + v_n \cos(\frac{\pi}{2} - \theta) \\ \dot{y} = v_t \sin\theta + v_n \sin(\frac{\pi}{2} - \theta) \end{cases}$$

Però, poichè il disco rotola senza strisciare sul piano, si ha che la velocità normale al verso di rotolamento è nulla: $v_n = 0$.

$$\begin{cases} \dot{x} = v_t \cos\theta \\ \dot{y} = v_t \sin\theta \end{cases}$$

Da cui si ottiene $\tan\theta = \frac{dy}{dx}$ ossia $dx\ \sin\theta = dy\ \cos\theta$

Il vincolo $\dot{x}\sin\theta - \dot{y}\ \cos\theta = 0$ viene indicato come vincolo sulla mobilità.

Per ogni ruota è possibile scrivere il vincolo anolonomo in termini di coordinate generalizzate q come

$$a(q)\dot{q} = 0$$

E per N ruote i vincoli posson esser scritti come

$$A(q)\dot{q} = 0$$

Tale espressione va ad indicare un vincolo Pfaffiano.

In definitiva i vincoli anolonomi non comportano una riduzione dello spazio delle configurazioni, ma una riduzione della mobilità istantanea del robot.

L'equazione sopra indica che ci sono delle velocità ammissibili, le quali possono essere generate da una matrice G(q), per cui si può scrivere il modello cinematico per un robot mobile con vincoli anolonomi nella forma $\dot{q} = G(q)v$, dove v è l'ingresso cinematico.

Questa rappresenta le direzioni di moto ammissibili nello spazio delle configurazioni ed è necessaria per affrontare le problematiche principali della robotica mobile:

- pianificazione di traiettoria;
- controllo;
- localizzazione.

Andando a considerare lo schema di principio di un robot mobile, esso può esser così rappresentato:

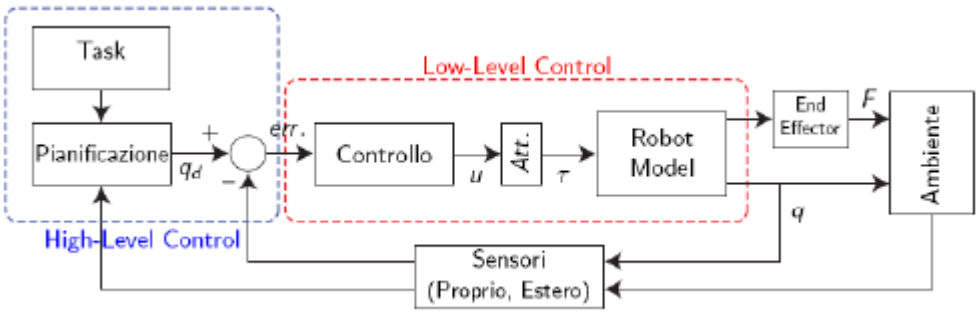

Fig. 3.3 Schema di principio di un robot mobile.

Si può considerare composto da:

- attuatori: motori DC con riduttori;
- end effector: pinza, ecc;
- sensori, a sua volta divisi in propriocettivi (encoders, giroscopi) ed eterocettivi (laser, visione);
- controllo, di alto e basso livello.

Il controllo di basso livello si occupa solamente di controllare gli attuatori del robot secondo le istruzioni del controllo ad alto livello. Controllori PI ad alto guadagno controllano i motori del robot affinchè si muova secondo il profilo di velocità desiderato; inoltre, se i guadagni sono abbastanza alti, il controllo di basso livello rende il robot un sistema puramente cinematico.

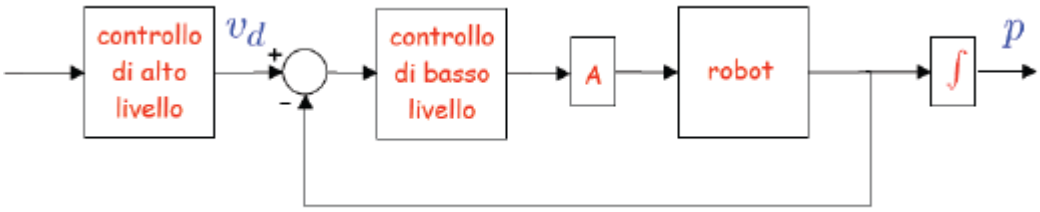

Fig. 3.4 schema di controllo a basso livello.

Parlando della schema di controllo ad alto livello (quello trattato in questa tesi,ad eccezione della pianificazione di traiettoria), a seconda della configurazione di set point (valore di riferimento che si va ad impostare per una data variabile), questo deve decidere che velocità imporre al robot. Dal punto di vista di questo tipo di controllo, il rover è un sistema puramente cinematico, con ingressi in velocità.

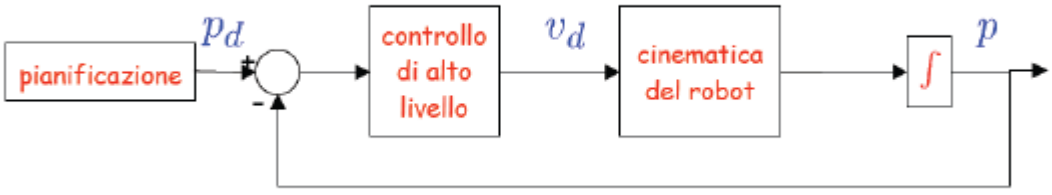

Fig. 3.5 Schema di controllo ad alto livello.

In definitiva, il controllo di alto livello si occupa di decidere il movimento mentre quello a basso livello si occupa di controllare i motori affinchè tale movimento sia attuato.
Della parte al alto livello si include anche la pianificazione della traiettoria da seguire (non trattata nel mio lavoro di tesi, in quanto si deve fare riferimento al gruppo che si occupa della navigation); questa ha la seguenti caratteristiche:

- fornisce il set point al controllo di alto livello;
- deve tenere in considerazione i vincoli sulla mobilità del robot.

Strettamente connessa alla pianificazione di traiettoria, c'è la navigazione.
Infatti, nel caso in esame, il rover si muoverà in un ambiente in cui sono presenti ostacoli, fissi o mobili; il robot, tramite la conoscenza dell'ambiente circostante e tramite l'utilizzo di sensori a bordo deve riconoscere dove sono gli ostacoli ed evitarli al fine di navigare senza collisioni all'interno dello spazio di lavoro.
D'altra parte il progetto del controllo di basso livello:

- è semplice in quanto si riduce al progetto di un controllore PI per un motore elettrico;
- non risente dai vincoli introdotti dalle ruote.

La parte d'alto livello invece:

- richiede di considerare il modello cinematico del rover;
- generalmente è complesso.

Si prosegue con una piccola descrizione delle altre parti che entrano in gioco nel controllo di un robot mobile, cominciando dalla pianificazione.
Essa consiste nel problema di determinare una traiettoria nello spazio delle configurazioni per portare il rover da una certa configurazione iniziale ad una finale ammissibile (ossia compatibile coi vincoli cinematici del robot). Di conseguenza, ogni punto della traiettoria dev'essere compatibile coi vincoli cinematici del robot.
La traiettoria può esser decomposta in un cammino q(s) e una legge oraria s = s(t); il primo rappresenta la forma geometrica della curva parametrizzata da s, mentre la legge oraria va a rappresentare il modo in cui la curva viene percorsa.
Andiamo a fare una separazione spazio temporale della traiettoria : $\dot{q} = \frac{dq}{dt} = \frac{dq}{ds}\dot{s} = q'\dot{s}$

Affinchè la traiettoria sia ammissibile per il robot, essa deve soddisfare i suoi vincoli anolonomi, per cui $A(q)\dot{q} = A(q)q'\dot{s} = 0$, però $\dot{s} > 0$, quindi infine si ottiene la condizione di ammissibilità geometrica del cammino, che si esprime nella forma $A(q)q' = 0$.
Un cammino ammissibile sarà dato da $q' = G(q)\tilde{u}$.
Il cammino geometrico si trova determinando gli ingressi geometrici ũ; una volta trovato quello ammissibile, occorrerà scegliere una legge oraria s(t), ovvero la velocità con cui il cammino è percorso, per determinare completamente la traiettoria.

Data una traiettoria o una configurazione desiderata, il passo successivo consiste nel costruire una legge di controllo che porti il rover a seguire la traiettoria oppure nella configurazione desiderata
Il problema del controllo del moto di un robot mobile viene affrontato utilizzando il modello cinematico e si va a supporre che gli ingressi cinematici agiscano direttamente sulle variabili di configurazione. Spesso gli ingressi di controllo sono v e ω , infatti molte volte non è possibile imporre direttamente la coppia motrice sulle ruote a causa di anelli di controllo a basso livello integrati nell'architettura hardware o software.
Parlando del controllo del moto, si possono seguire due tipi diversi di strategie:

- regolazione di configurazione: il robot deve raggiungere una configurazione desiderata $q_d = (x_d, y_d, \theta_d)^T$ a partire da una configurazione iniziale $q_0 = (x_0, y_0, \theta_0)^T$. Questo rappresenta il problema più difficile;
- inseguimento di traiettoria: il robot deve riprodurre asintoticamente una traiettoria cartesiana desiderata $(x_d(t), y_d(t))$ a partire da una configurazione iniziale $q_0 = (x_0, y_0, \theta_0)^T$ che può essere o meno agganciata alla traiettoria. Questo è il problema più semplice e di maggior interesse pratico.

Si procede quindi nel dare una breve introduzione dell'inseguimento di traiettoria, e più precisamente del caso Input-Output state feedback linearization.
Questo dipende dal punto preso come riferimento del robot (uscita del sistema), ossia dal punto al quale vogliamo assegnare il moto desiderato. Se andiamo a prendere il centro dell'asse come uscita da controllare si hanno dei vincoli nella mobilità del robot, in quanto questo non può mai avere una velocità laterale rispetto all'orientamento del veicolo.

Si può, quindi, andare a prendere un punto fuori dall'asse delle ruote come uscita da controllare, avente coordinate $\begin{cases} x_B = x + b\cos\theta \\ y_B = y + b\sin\theta \end{cases}$ e tale punto non ha vincoli cinematici e può muoversi, istantaneamente anche lateralmente rispetto la direzione del moto.

Si possono quindi definire due ingressi cinematici che determinano la velocità lungo l'asse x e y del punto B. Data una traiettoria (x_{des}, y_{des}) da inseguire, è possibile trovare la due velocità che garantiscono l'inseguimento usando delle semplici leggi lineari del tipo $\begin{cases} v_{dx} = \dot{x_{des}} + k_1(x_{des} - x_B) \\ v_{dy} = \dot{y_{des}} + k_2(y_{des} - y_B) \end{cases}$

che, ponendo $e_x = x_{des} - x$ e $e_y = y_{des} - y$, si ottengono $\begin{cases} \dot{e_x} + k_1 e_x = 0 \\ \dot{e_y} + k_2 e_y = 0 \end{cases}$.

Da questi è poi possibile determinare gli ingressi cinematici v e ω.

Dopo aver parlato della pianificazione e dell'inseguimento di traiettorie si va ora a dare una descrizione riguardo la navigazione (quella autonoma è uno dei vari task della competizione).

Spesso un robot si trova a doversi muovere in un ambiente in cui sono presenti ostacoli (fissi o mobili); esso, tramite la conoscenza dell'ambiente e/o tramite l'utilizzo dei sensori a bordo, deve riconoscere dove sono gli ostacoli ed evitarli al fine di navigare in maniera sicura, ossia senza collisioni, all'interno dello spazio di lavoro. Da questo si può dire che la navigazione indica il problema di trovare un percorso libero da collisioni che porti il rover dalla configurazione di partenza a quella di arrivo (entrambe note).

Una delle strategie utilizzate per la navigazione viene indicata col nome di bug algorithms.

Questi si basano sulla possibilità del rover di percepire, tramite opportuni sensori eterocettivi, la presenza di un ostacolo quando vi sono molto vicini o addirittura quando vi sono a contatto.

L'idea di base è la seguente: muovere il rover in linea retta lungo l'obiettivo; quando si incontra un ostacolo lo si circumnaviga fino a quando non è possibile muoversi in linea retta verso l'obiettivo.

Si va a considerare il robot come un punto dotato di sensore di contatto e si suppone che sia in grado di conoscere la propria posizione, oltre a quella della configurazione di partenza e obiettivo.

Esso può superare l'ostacolo in vari modi, ne elenchiamo qui solo alcuni:

- col primo metodo, il robot può assumere due tipi di comportamento tra i quali commuta; il primo, detto motion to goal, consiste nel movimento del rover (a partire da un leave point) lungo la linea che connette il leave point all'obiettivo finchè non lo raggiunge oppure incontra un ostacolo. Se si incontra un ostacolo il punto di contatto viene detto hit point ed il comportamento del robot commuta al boundary following. Qui, a partire dal punto di contatto, il robot circumnaviga l'intero ostacolo fino a ritornare all'hit point, poi determina il punto più vicino alla retta per l'obiettivo e si sposta lungo il perimetro dell'oggetto fino a raggiungerla. Questo punto viene detto leave point ed il comportamento del rover ritorna al motion to goal.
- col secondo metodo, il robot può assumere gli stessi due comportamenti del metodo sopra, ma sono lievemente differenti. Infatti il motion to goal è analogo al caso sopra, mentra il boundary following segue un comportamento diverso, in quanto il robot circumnaviga l'ostacolo fino a trovare nuovamente la retta, il punto in cui si ricongiunge con essa viene detto leave point e il comportamento del rover ritorna in motion to goal.

  Si nota che il primo metodo compie una ricerca esaustiva e trova il punto migliore, però richiede che venga percorso l'intero perimetro dell'ostacolo; invece il secondo metodo usa un approccio opportunistico.
- navigazione attraverso funzioni potenziali; con questo metodo è possibile pianificare il moto anche per sistemi con variabili di configurazione di dimensione elevata.

  Quest'approccio guida il rover come se fosse una particella immersa in un campo potenziale dato dai gradienti di svariate funzioni potenziali. Alla configurazione obiettivo è associata una funzione potenziale che genera un gradiente che attira la particella mentre ad ogni ostacolo è associato una funzione potenziale che genera un gradiente che respinge le particelle. La combinazione dei due gradienti produce un percorso libero da collisioni verso la configurazione obiettivo.

Per poter seguire la traiettoria desiderata, un rover deve conoscere la propria configurazione e per poter eseguire una strategia di navigazione non banale deve conoscere una mappa dell'ambiente circostante.

In più ogni controllore in retroazione richiede la conoscenza dello stato del robot (in termini di posizione assoluta). In generale, quindi, per rendere veramente autonomo un rover, è necessario che esso sia in grado di localizzarsi e di costruire una mappa dell'ambiente circostante.
Per fare questo si può utilizzare lo SLAM (Simultaneous Localization And Mapping).
Per descrivere lo SLAM possiamo partire dall'odometria, la quale è la tecnica per stimare la posizione di un veicolo su ruote basandosi su informazioni provenienti da sensori che misurano lo spazio percorso da alcune delle ruote e l'angolo di sterzo (se presente).
Gli encoder incrementali sulle ruote ne misurano la rotazione, ma non l'assetto del robot in un sistema di riferimento fisso assoluto. Se una ruota slitta, l'encoder rileva uno spostamento che non è coerente col cambio d'assetto del rover.
L'uso della sola odometria porta quindi ad errori di localizzazione che vanno ad accumularsi durante il moto del rover; per compensare tali errori e per costruire una mappa dell'ambiente circostante si usano sensori eterocettivi (scanner laser, telecamere, GPS, sonar), solidali al rover e capaci di misurare la posizione del robot rispetto all'ambiente esterno.
Da cui si può dire che lo SLAM funziona a partire dalla conoscenza delle misure effettuate per determinare la posizione del veicolo e costruire la mappa delle posizioni dei landmark (punti di riferimento). Localizzazione e mapping sono problemi correlati ed una soluzione è ottenibile solamente considerando i due problemi insieme.

3.2 La configurazione skid-steering e il modello cinematico per il rover Morpheus

Rover che utilizzano la configurazione skid-steering sono ampiamente utilizzati, per la loro semplicità a livello di meccanismi e robustezza.
Questo tipo di configurazione si basa sul controllo delle velocità dei gruppi di ruote destre e sinistre.
Alcune delle sue caratteristiche principali sono:

- elevata manovrabilità;
- la struttura meccanica semplice e robusta (questo permette di poter disporre di più spazio per alloggiare attrezzature nel rover);

- una buona mobilità su molti tipi di terreni.

Tuttavia, a causa delle complesse interazioni tra terreno e ruote e dei vincoli cinematici, risulta decisamente complicato arrivare a capire i modelli cinematico e dinamico di tali robot e a poter descrivere con accuratezza il moto.
Infatti, in questo tipo di locomozione di fondamentale importanza è il fenomeno dello slittamento.
Il movimento di un rover skid-steering si ottiene mediante l'applicazione di velocità ai gruppi di ruote dai lati destro e sinistro. Sebbene questa tipologia di locomozione porti numerosi benefici, dall'altra parte il controllo è molto complesso, in quanto le ruote devono slittare lateralmente per seguire una traiettoria curvilinea.
Se la proiezione del centro d'istantanea rotazione (CIR) del veicolo lungo l'asse longitudinale diventa grande, il rover può perdere la stabilità di moto come risultato dello slittamento.
In generale, il controllo di un tale sistema necessita non solo del modello cinematico, ma anche di quello dinamico.
L'utilizzo del modello cinematico è necessario per le operazioni di calcolo a bordo (della UDOO) in ambito di dead-reckoning e per il controllo del moto nel breve periodo.

Andiamo ora a considerare la procedura utilizzata per la determinazione del modello cinematico del rover Morpheus.
Nel formulazione del modello cinematico del rover sono state fatte alcune assunzioni al fine di ottenere un primo modello relativamente semplice:

- si considera un terreno piano, questo permette di dire che i bilancieri non oscillano e ciò implica che le ruote rimangono alla stessa distanza dal centro;
- le ruote sono sempre in contatto col terreno e non c'è slittamento;
- non si considera lo slittamento laterale;
- poiché l'asse centrale delle ruote è posto a pari distanza dagli altri due, si può considerare l'origine del sistema di riferimento solidale al rover come centrato su quell'asse e posto a metà della sua lunghezza.

Queste assunzioni permettono di dire che il modello cinematico del rover è uguale a quello di un rover con guida differenziale e, poiché l'asse centrale è posto a pari distanza dagli altri due e contiene l'origine del sistema di riferimento solidale, che esso è uguale a quello di un rover con 4 ruote.

Si procede ora con la determinazione del modello cinematico di un rover a 4 ruote (nelle ipotesi fatte esso è uguale a quello del rover a 6).

Per determinarlo si assume che il robot sia posto su una superficie piana e con un sistema di riferimento ortonormale fisso (X_g, Y_g, Z_g). Questo si posiziona con l'origine in comune al sistema solidale al rover definito dalla terna (x_l, y_l, z_l).

Le direzioni dei versori che formano la terna solidale col rover sono le seguenti:

- asse x diretto lungo la direzione di avanzamento del rover;
- asse z rivolto verso alto;
- asse y che completa la terna destrogira.

Successivamente, quando il lavoro del team GPS sarà concluso, si collegherà questo sistema fisso ma non inerziale con un sistema compatibile con il GPS e le sue misure.

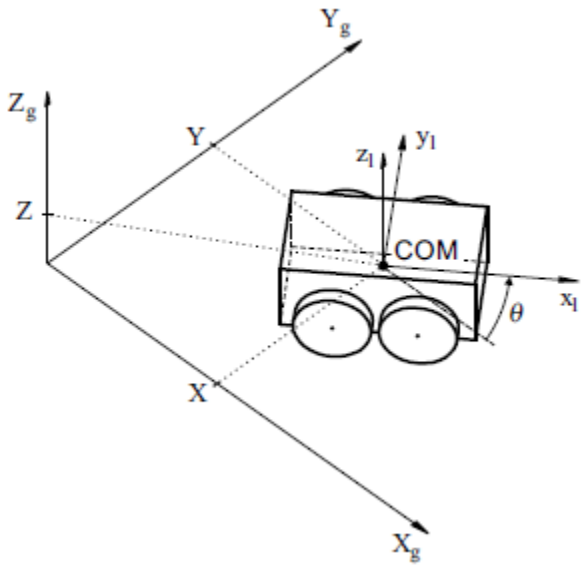

Fig. 3.6 Immagine dei sistemi di riferimento del rover. Qui non hanno l'origine comune per rendere più chiara la figura.

Poiché si considera il moto piano, nella trattazione si assumerà la coordinata Z come costante.

Si supponga che il rover si muovi nel piano con velocità lineare espressa nel sistema locale come $v = [v_x \; v_y \; 0]^T$ e ruoti con un vettore velocità angolare $\omega = [0 \; 0 \; \omega]^T$.

Si definisca il vettore contenente le coordinate generalizzate $q = [X \; Y \; \theta]^T$ (X e Y sono le coordinate di un sistema di riferimento rispetto l'altro, mentre θ rappresenta l'orientamento di un sistema rispetto l'altro) ed il vettore delle velocità generalizzate $\dot{q} = [\dot{X} \; \dot{Y} \; \dot{\theta}]^T$.

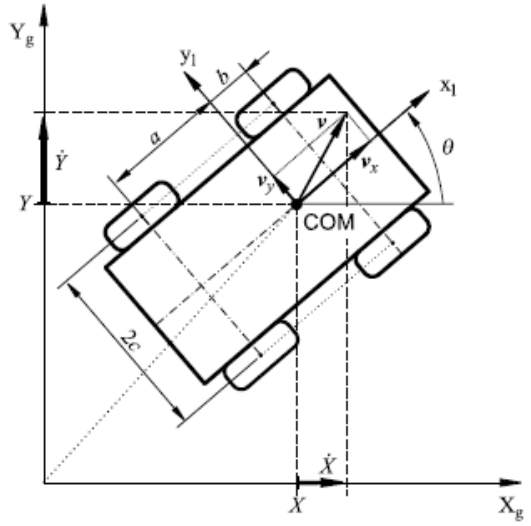

Fig. 3.7 Rapprentazione nel piano x-y. (quest'esempio pone l'origine nel centro di massa ma ciò non porta a differenza dal punto di vista della cinematica).

Dalla fig. 3.7 si può notare che $\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$.

Inoltre, sapendo che il moto è piano, si può scrivere $\dot{\theta} = \omega$.

L'equazione sopra non impone nessuna restrizione al movimento del rover nel piano, poiché descrive esclusivamente la cinematica del corpo libero. Perciò è necessario analizzare la relazione tra la velocità delle ruote e le velocità locali.

Si supponga che la i-esima ruota si muova con velocità angolare $\omega_i(t)$, con i = 1, 2,.., 6, e ciò può anche esser preso come l'input di controllo.

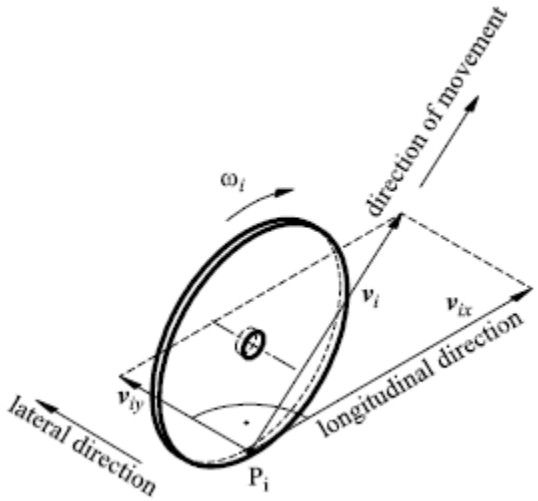

Fig. 3.8 Velocità di una ruota.

Per semplicità, si trascura lo spessore della ruota e si assume, quindi, che sia in contatto col piano nel punto P_i come si vede in fig. 3.8.

A differenza della maggior parte dei veicoli dotati di ruote, generalmente in questa situazione la velocità laterale, v_{iy}, è diversa da zero, e questa proprietà deriva direttamente dalla struttura meccanica del rover, la quale rende necessario lo slittamento laterale per poter ottenere un cambio d'orientazione dei veicolo. Per questo le ruote sono tangenti alla traiettoria solo se la velocità angolare è nulla, ossia quando il rover si muove in linea retta.

Tuttavia, in questa situazione si analizza un modello semplificato, per cui si va a trascurare lo slip longitudinale tra le ruote ed il terreno.

In base a quanto detto sopra si può di conseguenza ottenere che $v_{ix} = r_i\omega_i$, dove v_{ix} rappresenta la componente longitudinale del vettore velocità $\mathbf{v}_i$ dell'i-esima ruota espressa nel sistema locale di coordinate, e r_i rappresenta il raggio dell'i-esima ruota.

Per sviluppare il modello cinematico bisogna considerare tutte le ruote assieme.

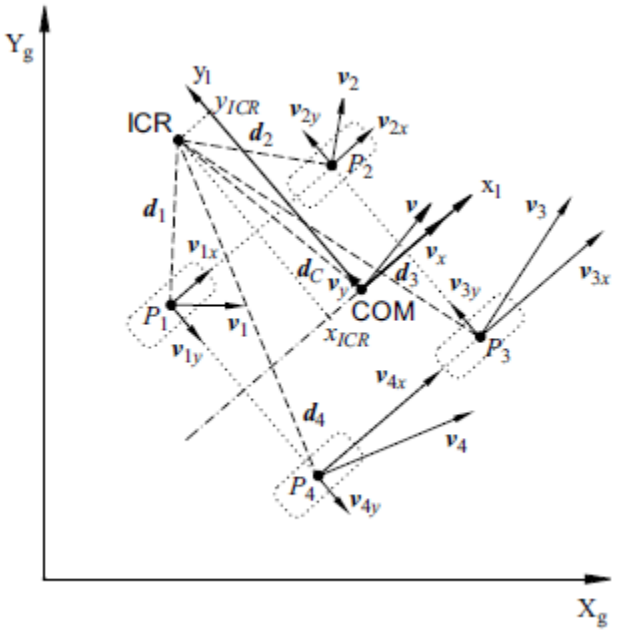

Fig. 3.9 Velocità di tutte le ruote.

Dalla figura 3.9 si possono determinare i vettori $d_i = [d_{ix} \quad d_{iy}]^T$ e $d_C = [d_{Cx} \quad d_{Cy}]^T$, i quali sono definiti dal centro istantaneo di rotazione al sistema di riferimento locale.

Di conseguenza, basandosi sulla geometria della fig. 3.9, si può dedurre la seguente espressione:

$$\frac{\|v_i\|}{\|d_i\|} = \frac{\|v\|}{\|d_C\|} = |\omega|$$

o, in forma più dettagliata,

$$\frac{v_{ix}}{-d_{iy}} = \frac{v_x}{-d_{Cy}} = \frac{v_{iy}}{d_{ix}} = \frac{v_y}{d_{Cx}} = \omega$$

dove il simbolo sopra indica la norma Euclidea.

Definite le coordinate del CIR (centro istantanea rotazione) nel sistema locale

$$CIR = (x_{CIR}, y_{CIR}) = (-d_{Cx}, -d_{Cy})$$

è possibile scrivere la velocità angolare nel modo seguente

$$\frac{v_x}{y_{CIR}} = -\frac{v_y}{x_{CIR}} = \omega$$

Si ha inoltre che le coordinate del vettore d_i soddisfano le seguenti relazioni

$$d_{1y} = d_{2y} = d_{Cy} + c$$
$$d_{3y} = d_{4y} = d_{Cy} - c$$
$$d_{1x} = d_{4x} = d_{Cx} - a$$
$$d_{2x} = d_{3x} = d_{Cx} + b$$

Dove a, b e c sono parametri cinematici positivi del robot (si ha anche che, essendo i 3 assi delle ruote equidistanziati, a = b).
Combinando le varie espressioni si arriva a

$$v_L = v_{1x} = v_{2x}$$
$$v_R = v_{3x} = v_{4x}$$
$$v_F = v_{2y} = v_{3y}$$
$$v_B = v_{1y} = v_{4y}$$

Si ha che v_L e v_R indicano le coordinate longitudinali delle velocità della ruota sinistra e destra, mentre v_F e v_B indicano le coordinate laterali della velocità delle ruote anteriori e posteriori.
Comunque in questo caso non si considera diversa la velocità anteriore e posteriore, in quanto in questo modello si assume le velocità come gruppo di ruote di destra e di sinistra.
Andando inoltre a porre che tutti i raggi delle ruote siano uguali e pari ad r, si può andare a scrivere la seguente relazione

$$\omega_w = \begin{bmatrix}\omega_L \\ \omega_R\end{bmatrix} = \frac{1}{r}\begin{bmatrix}v_L \\ v_R\end{bmatrix}$$

dove le due velocità angolari vanno ad indicare le velocità dei gruppi di ruote sinistra e destra, rispettivamente.
Infine è possibile ricavare un'espressione che lega le velocità del robot con quelle delle ruote

$$\eta = \begin{bmatrix} v_x \\ \omega \end{bmatrix} = r \begin{bmatrix} \frac{\omega_L + \omega_R}{2} \\ \frac{-\omega_L + \omega_R}{2} \end{bmatrix}$$

Da quest'equazione è possibile ricavare le relazioni inverse, che sono quelle a noi necessarie, in quanto le nostra variabili di ingresso sono le velocità del rover mentre le incognite sono le velocità da imporre ai due gruppi di ruote.

Cosi in definitiva si può ricavare

$$\omega_R = \frac{1}{r}\,(v_x + c\omega)$$
$$\omega_L = \frac{1}{r}\,(v_x + c\omega)$$

Per completare il modello cinematico del rover bisogna andare ad imporre il seguente vincolo sulla velocità

$$v_y + x_{CIR}\dot{\theta} = 0$$

Quello appena introdotto è un modello cinematico relativamente semplice.
Nella realtà, la presenza di un terreno accidentato (che presenta dossi ed avvallamenti) porta ad un'inclinazione dei bilancieri, in quanto devono sempre assicurare il contatto delle ruote col terreno.
Questo, però, porta le ruote, ed in particolare quelle connesse al bilanciere posteriore, a variare la propria distanza rispetto l'origine del sistema di riferimento; ciò porterà al fatto che le ruote dovranno avere una diversa velocità angolare.

Inoltre il fatto che il veicolo ruoti per effetto dello slittamento laterale qui non viene considerato.
Il fatto che si abbiano basse velocità in gioco fa si che quest'approssimazione non sia del tutto errata, però inevitabilmente porterà ad un errore.
Per compensare questo fatto bisognerà introdurre anche il modello dinamico, cosa che in questa tesi non è stata trattata.
Per migliorare ancora il modello sarebbero stati necessari dei test sulla locomozione del rover, fatto indispensabile per ottenere alcuni parametri che vanno a caratterizzare il modello dinamico ma, per il fatto che microcontrollori e motori non sono ancora disponibili, ciò non si è potuto fare.

CAPITOLO 4

Algoritmi per la movimentazione del rover

4.1 Introduzione

Il seguente capitolo rappresenta il cuore di questo progetto di tesi, in quanto qui si sono andati a studiare e ad analizzare vari algoritmi per la definizione delle velocità angolari da imporre ai gruppi destro e sinistro di ruote.

Il capitolo inizia con una breve descrizione del linguaggio di programmazione C++, il quale è stato utilizzato per la scrittura dei diversi codici.

Successivamente si andrà a descrivere la procedura utilizzata per sviluppare i programmi.

Il lavoro è stato diviso in due fasi:

- nella prima fase si sono andati a scrivere due codici, uno in C e uno in C++; infatti si voleva che fossero sfruttate entrambe le CPU della UDOO, in quanto si volevano uscite analogiche;
- nella seconda fase si è andato a definire un unico codice in C++, in quanto, non essendoci più la necessità di avere uscite analogiche si è preferito riunire tutto l'algoritmo in un unico codice.

Questa fase di scrittura degli algoritmi ha riscontrato diversi problemi, in quanto il sistema operativo utilizzato è Linux, mentre la programmazione classica viene implementata in Windows. Questo ha portato ha dover modificare i primi algoritmi, implementati in Windows per semplicità e maggior conoscenza dell'ambiente, per poterli adattare alle particolarità tipiche del sistema operativo installato sulla scheda UDOO.

Per la visualizzazione completa degli algoritmi completi si rimanda all'appendice.

4.2 Introduzione al linguaggio C++

Il linguaggio di programmazione C++ rappresenta un'estensione del linguaggio di programmazione C e, per questo motivo, i compilatori C++ possono compilare anche i programmi in C.

Le principali estensioni sono:

- miglioramenti rispetto al linguaggio C;
- supporto per la gestione dei tipi di dato astratto;
- supporto per la programmazione orientata agli oggetti.

Andando ad analizzare la caratteristiche del linguaggio si può notare che:

- esso ha diverse componenti, distinguibili in direttive (iniziano col simbolo #), definizioni (di variabile o altro), funzioni (tra cui la funzione speciale main) e classi;
- la direttiva *#include <iostream.h>* include la libreria delle funzioni predefinite di input/output tipiche del C++ (questo valido per Windows, infatti cambiando il sistema operativo cambiano anche le librerie);
- le parentesi graffe delimitano un blocco, ciò che è dichiarato all'interno di uno di questi è visibile solamente all'interno del blocco stesso;
- i commenti possono esser specificati in due modi, o racchiusi tra /* e */, oppure con // , però quest'ultimo continua fino alla fine della riga.

Le variabili vengono dichiarate scrivendo il tipo di variabile seguito dal suo nome.

Una funzione viene specificata scrivendo, nell'ordine, il tipo restituito, il nome della funzione, la lista dei parametri formali ed un blocco di istruzioni.

Quando si va ad invocare una funzione si devono specificare i parametri attuali che devono corrispondere in numero, ordine e tipo a quelli formali.

Ogni blocco d'istruzioni deve concludersi con un ; .

Tra le varie funzioni una funzione speciale è la *main()*, che è quella che viene eseguita al lancio del programma.

Dopo aver citato alcune delle caratteristiche del linguaggio, andiamo a descrivere le espressioni e le istruzioni.

Le prime possono essere delle costanti, delle variabili oppure un'espressione composta (formata da un operatore e da uno o due comandi che a loro volta sono espressioni).
Un'espressione seguita da ; indica un'istruzione semplice, mentre una composta è una sequenza di istruzioni racchiuse dai simboli { e }.
Tra le istruzioni più utilizzate ci sono:

- istruzioni condizionali: tra queste citiamo le if-else e le switch-case;
- cicli: questi possono esser suddivisi in cicli while, do-while e for.

L'istruzione *switch-case* viene utilizzata per evitare di utilizzare if troppo concatenati e consiste nel valutare un'espressione e dividere l'azione a seconda dei valori che tale espressione estrae; la sintassi è la seguente:

switch (espressione)
{
case valore1 :statement1; (oppure un blocco{})
…..
case valoreN :statementN;
case default :statement; (se nessun caso è vero)
}

Si procede con la descrizione sintetica alcuni di questi cicli, che si rivelano essere molto importanti.
Il ciclo while si presenta nella forma *while(condizione){…}*, nella quale il blocco {…} viene eseguito finchè la condizione iniziale è vera.
Se si desidera fare un loop prima che sia finito l'intero set di statements nel blocco while si inserisce *continue*, col quale torno alla testa del loop.
Se invece si desidera uscire dal loop prima che sia finita la condizione utilizzo break.
Nel ciclo *do{…}while* , invece, l'esecuzione del blocco (o del singolo statement) è garantita almeno una volta.

I tipi di dato (necessari nella dichiarazione delle variabili) sono essenzialmente due: quelli primitivi principali (int, short, long, float, double, long double, char, bool) e quelli enumerati (enum).
In una variabile risulta importante anche il suo scope, ossia la sua visibilità, che indica la porzione del codice in cui tale variabile dichiarata è utilizzabile. Come regola si ricorda che lo scope di una variabile (o di un qualsiasi altro identificatore) si estende dal punto immediatamente successivo la dichiarazione fino alla fine del blocco d'istruzioni (che è delimitato dalla parentesi graffe) in cui essa è inserita.
Una seconda regola dice che all'interno dello stesso blocco non si può dichiarare più volte lo stesso identificatore, però può essere ridichiarato in un blocco annidato; in questa situazione la nuova dichiarazione nasconde quella più esterna che ritorna visibile una volta che si è usciti dal blocco in cui si è ridichiarato l'identificatore. Fanno eccezione le variabile globali, le quali sono visibili a livello di file.
Il tipo di dato può essere convertito, e ciò è possibile implicitamente tramite un'assegnazione oppure esplicitamente tramite l'operazione di "cast".
Detto questo si può andare a definire un termine costante semplicemente inserendo il termine *const* prima della definizione di una variabile; tale costante va inoltre inizializzata.
È possibile fare dei riferimenti ad una variabile già definita (ciò si indica con alias) e per fare ciò basta inserire nella dichiarazione il carattere &.
Con il tipo struct è possibile definire dati strutturati e per questi si possono definire delle operazioni da effettuare su di esso.

Un variabile molto importante nell'ambiente C++ è il puntatore, ossia una variabile che contiene un riferimento ad un'altra variabile. Da un punto di vista fisico un puntatore è l'indirizzo di una locazione di memoria (RAM). Per dichiararlo si inserisce il simbolo * prima del nome di una variabile.
Se, inoltre, si applica l'operatore & ad una variabile questo restituisce il puntatore ad essa.

Un'altra possibilità che offre il linguaggio è quella di poter creare (allocare) variabili e oggetti in maniera statica o dinamica. Nella prima situazione, una variabile esiste (è utilizzabile) dal

momento della sua dichiarazione fino al termine del blocco in cui essa è stata dichiarata (scope della variabile).
Nella seconda, invece, la variabile viene allocata e deallocata esplicitamente con opportuni operatori (new e delete); la gestione avviene, operativamente, mediante l'utilizzo di puntatori e della costante simbolica predefinita NULL. L'allocazione dinamica viene principalmente usata per la gestione di collezioni di dati la cui cardinalità non è nota a priori.

Durante l'esecuzione di un programma (runtime) la memoria viene gestita dal sistema in due modi:

- gestione statica: viene allocata dal sistema operativo un'area di memoria di dimensione fissa per tutta l'esecuzione del programma (a questo si possono collegare le variabili globali);
- gestione dinamica: vengono allocate due aree di memoria di dimensioni variabili che vengono usate in modo diverso. Il primo di questi viene detto stack, dove, in un record di attivazione, vengono allocate automaticamente tutte le variabili locali ed i parametri attuali quando una funzione viene invocata. Successivamente , quando l'esecuzione della funzione termina, il record d'attivazione viene cancellato e lo stack viene riportato nella situazione antecedente all'invocazione.

  Il secondo modo per la gestione dinamica viene chiamato heap, nel quale la gestione viene lasciata al programmatore tramite creazione e distruzione dinamica di variabili (tramite puntatori).

Si continua ora con la descrizione delle funzioni: esse sono a tutti gli effetti dei sottoprogrammi che vanno ad agire sui dati.
Come già detto prima, ogni programma presenta una funzione speciale, detta main.
Molto importante in C++ è la distinzione tra definizione e dichiarazione di una funzione; la definizione comporta la specifica completa della funzione (ossia la sequenza di istruzioni da svolgere), mentre la dichiarazione va a specificare solo le caratteristiche della funzione (numero e tipo dei parametri di ingresso e uscita) e viene anche indicata col nome di header.

Possono esserci funzioni con lo stesso nome ma differente lista di parametri di input (per numero o tipo) e return type, queste vengono chiamate polimorfe.

Infine si va a discutere dell'organizzazione dei programmi. Ogni programma in C++, generalmente, è distribuito su più file e ognuno di questi costituisce una componente (modulo) dell'applicazione.

Ogni modulo viene decomposto in due file:

- Le dichiarazioni vengono poste in file detti header che hanno estensione ".h"; con tale file si dichiarano i servizi offerti dal modulo;
- Le definizioni vengono poste in file che hanno estensione ".C" o ".cpp" e tali file costituiscono l'effettiva implementazione dei servizi.

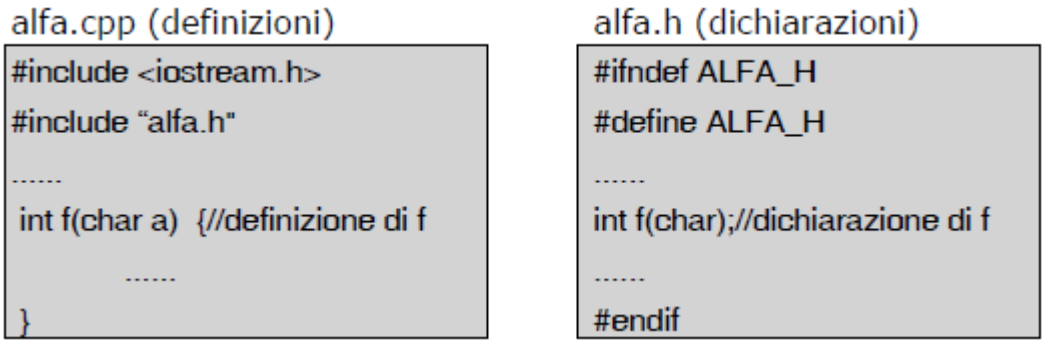

Fig. 4.1 Esempio dei due file in cui si divide un modulo.

È possibile anche la comunicazione tra componenti separate: infatti quando si vogliono usare in un file funzionalità (funzioni, dati e classi) definiti altrove, si introducono le loro dichiarazioni (contenute nei file header) e si rendono così visibili all'interno del file.

Per includere un file in un altro si usa la direttiva di compilazione:

- per le librerie di sistema si usa *#include* <...> ;
- per le librerie definite dal programmatore si utilizza *#include "..."*.

In questo modo si disaccoppia l'uso di una componente dalla sua effettiva implementazione e ciò permette la compilazione separata delle varie componenti.

Un programma comunica con l'esterno attraverso strutture logiche dette stream (o flussi) e questi possono essere associati a dispositivi fisici (video, tastiera, stampanti) oppure a file residenti in una memoria secondaria.

Le operazioni più comuni che si effettuano sugli stream sono quelle di lettura e scrittura, le quali possono essere:

- formattate, ossia prima del trasferimento convertono il contenuto di un flusso da una rappresentazione interna a dati tipati (e viceversa);
- non formattate, le quali non effettuano conversioni prima del trasferimento.

Ai dati di flusso si può accedere in maniera sequenziale, per cui si opera sempre sul dato successivo a quello su cui si è operato precedentemente, oppure diretta (o casuale), per cui si può operare su un dato qualunque del flusso.

In C++ esistono alcuni classi predefinite che gestiscono le operazioni di input ed output; queste, nella parte pubblica, contengono le operazioni disponibili su un flusso e, nella parte privata, una struttura detta buffer che memorizza blocchi di caratteri in transito tra la memoria principale ed il dispositivo (o file) associato al flusso.

Un flusso è un oggetto di una di queste classi predefinite ed è costituito (logicamente) da una sequanza di caratteri terminata da un carattere speciale (EOF).

Le classi predefinite per la gestione dell'I/O sono tutte derivate dalla classe base IOS che definisce il concetto astratto di flusso e tutte le operazioni di base su di esso.

Alcune delle classi più importanti per tale gestione sono:

- per le operazioni di I/O che coinvolgono dispositivi fisici si fa uso delle classi istream (ingresso) e ostream (uscita), definite nella libreria *iostream.h* ;
- per le operazioni di I/O, invece, che coinvolgono file si fa uso delle classi ifstream (ingresso), ofstream (uscita) e fstream(ingresso/uscita), definite nella libreria *fstream.h* .

Analizzando le operazioni di ingresso/uscita da/verso dispositivi fisici si fa uso di due flussi predefiniti, l'oggetto *cin* della classe istream che è associato al dispositivo standard d'ingresso (tastiera), e l'oggetto *cout* della classe ostream, il quale è associato al dispositivo standard di uscita (video).

Gli operatori formattati di base sono >> (ingresso) e << (uscita), i quali sono definiti per i tipi fondamentali, per i puntatori e per le stringhe.
Questi operatori restituiscono un riferimento ad un flusso e possono, quindi, essere composti; inoltre l'ordine di precedenza di questi è più alto solo degli operatori logici.
Andiamo a vedere altre funzioni d'ingresso non formattate:

- *int get()*: legge il prossimo carattere (qualunque sia) dal flusso d'ingresso e lo restituisce;
- *istream& get(char& ch)*: legge il prossimo carattere (qualunque sia) dal flusso di ingresso e lo memorizza in ch;
- *istream& getline(char& st, int n, char l='/n')*: legge una stringa di caratteri dal flusso d'ingresso e la memorizza in st; vengono letti n caratteri oppure finchè non si raggiunge il carattere l;
- *int gcount()*: restituisce il numero di caratteri effettivamente letti dall'ultima getline;
- *istream& read(9char* ps, int n)*: legge una sequenza di caratteri lunga n dal flusso d'ingresso e la memorizza a partire dall'indirizzo ps.

In uscita invece si possono avere le seguenti funzioni non formattate:

- *ostream& put(char ch)*: scrive il carattere ch (qualunque sia) nel flusso d'uscita;
- *ostream& write(const char* ps, int n)*: scrive n caratteri della stringa puntata da ps sul flusso d'uscita.

Per le operazioni di ingresso/uscita da/verso file è sufficiente definire un oggetto della classi ifstream, ofstream o fstream. Prima di operare su un file bisogna aprirlo utilizzando la funzione *open()* specificando il file fisico associato e la modalità di apertura:

- *ios::in* (apertura in lettura);
- *ios::out* (apertura in scrittura);
- *ios::in*|*ios::out* (apertura in lettura/scrittura).

Per default, un oggetto di ifstream viene aperto in lettura mentre uno di ofstream viene aperto in scrittura.
Per aprire un file di dati in scrittura, iniziando però dalla fine del file se esso è già presente su disco (append), è possibile utilizzando la scrittura *ios::app*.

4.3 Algoritmi

Si procede ora con l'analisi nel dettaglio degli algoritmi scritti per la determinazione delle velocità angolari delle ruote del rover a partire da input dati da tastiera.
Il paragrafo si dividerà in due parti:

- nella prima si vanno a descrivere due codici, il primo scritto in C++, mentre il secondo scritto utilizzando C;
- nella seconda si andrà a descrivere il secondo codice, scritto esclusivamente in C++.

Durante il periodo in cui la tesi è stata svolta, il codice è stato modificato per potersi adattare ad alcune modifiche richieste dal progetto Morpheus.
I problemi principali riscontrati sono stati il dover passare da una conoscenza delle librerie di C++ di Windows al dover utilizzare le librerie scritte per il sistema Ubuntu.
Molto spesso tali librerie, anche se definite come equivalenti tra un sistema operativo e l'altro, in realtà hanno portato a dover modificare l'idea iniziale che si aveva riguardo le scrittura del programma: infatti, molti dei problemi che a livello di codice in Windows sono stati risolti, in Linux non hanno una soluzione e, di conseguenza, si è dovuta attuare una diversa strategia.

Durante la programmazione su Windows si è utilizzato il compilatore Dev-C++, un IDE gratuito per la programmazione in C/C++. Un'IDE (Integrated Development Environment, ossia ambiente di sviluppo integrato) è un software, che durante la fase di programmazione, aiuta i programmatori nello sviluppo del codice sorgente di un programma.
L'IDE segnala errori di sintassi del codice direttamente in fase di scrittura, oltre ad avere tutta una serie di strumenti e funzionalità di supporto alla fase di sviluppo e debugging (individuazione degli errori, detti bug).

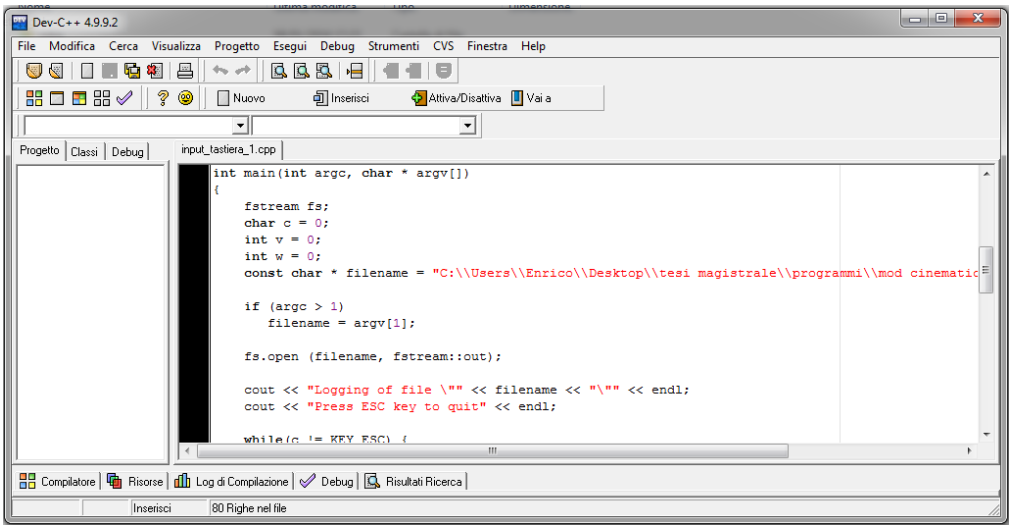

Fig. 4.2 La schermata di Dev-C++.

Successivamente, sul sistema operativo Ubuntu, è stato sviluppato il nuovo codice e si è andato a testarne il funzionamento prima di utilizzarlo nella scheda UDOO.

Su questo sistema operativo si è utilizzato un IDE diverso (in quanto Dev-C++ è supportato solo da Windows), Code::Blocks, uno strumento open source e multipiattaforma caratterizzato da un'architettura basata sui plugin (programmi non autonomi che interagiscono con un altro programma per ampliarne/estenderne le funzionalità originarie).

All'interno del sistema operativo della scheda elettronica non è stato necessario dover installare nuovi compilatori, in quanto, una volta completati la scrittura del codice e la fase di debugging, è sufficiente lanciare l'eseguibile del programma da terminale per poterlo eseguire.

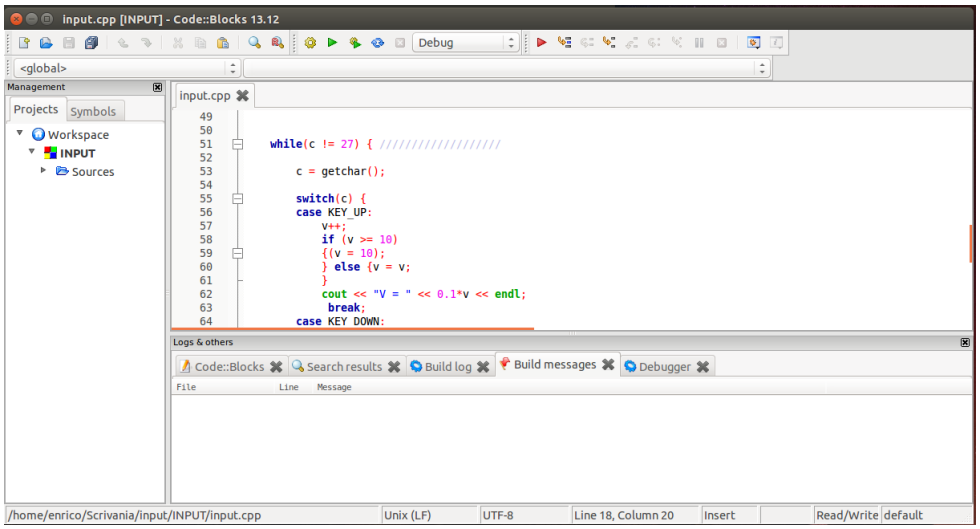

Fig. 4.3 La schermata di Code::Blocks.

4.3.1 I due codici separati

All'inizio si volevano collegare i driver dei motori direttamente alla scheda UDOO, senza i microcontrollori, e ciò ha portato al problema dell'avere installate nella scheda solo due uscite di tipo analogico. Si è cosi deciso di scrivere una parte di codice utilizzando C++ e di lanciarla sul processore principale (i.MX6), mentre la seconda parte (scritta in linguaggio C per meglio potersi poi adattare al linguaggio di Arduino) andava a prendere la uscite del primo codice e a rielaborlarle sul processore compatibile con Arduino 2, per poi inviare gli output ai motori tramite le uscite analogiche di Arduino.

Il primo codice, essenzialmente, attende da parte dell'operatore un input da tastiera dato dalle frecce direzionali; questi comandi da tastiera generano due velocità, una lineare ed una angolare, le quali vengono inviate, tramite collegamento tra i due processori (come visto nel capitolo 2), al secondo codice. Inoltre, salva su file la lista dei comandi premuti, così da avere uno strumento con cui poter ricostruire la traiettoria seguita dal rover.

Il secondo codice va a prendere come dati in input le due velocità inviate dall'altro codice ed, utilizzando le equazioni della cinematica ricavate alla fine del capitolo 3, va a determinare la velocità angolare da applicare al gruppo di ruote destra e sinistra.

Andiamo ora a descrivere nel dettaglio i due codici, cominciando con quello scritto in C++.
Prima del programma si vanno ad inserire tre *#include*, ossia delle direttive per il preprocessore. Queste sono succedute da un nome racchiuso da < >, e tale nome va ad identificare un file che il preprocessore va a cercare e a copiare (senza interpretarlo) nel file di testo che viene poi passato al compilatore. Grazie al lavoro del processore il compilatore, durante il controllo della sintassi del codice preparato dall'operatore, si trovano automaticamente le dichiarazioni di tutte le funzioni usate dentro il file sorgente.
Le tre direttive usate sono:

- *#include <conio.h>* : qui sono dichiarate diverse funzioni create perché un programma possa eseguire particolari operazioni di input/output su console;
- *#include <iostream>* : permette di leggere e scrivere su di un flusso in input/output;
- *#include <fstream>* : l'equivalente della direttiva sopra ma utilizzata per i file.

Successivamente si va a scrivere *using namespace std*, che indica che si sta usando lo spazio standard dei nomi.
Dopo si vanno ad inserire la varie costanti, inserite precedute dalla direttiva *#define*; in questo caso come costanti si assumono i valori assunti dai tasti occupati dalle frecce direzionali e dall'esc, necessario per uscire dal programma.
L'utilizzo della direttiva *define* diminuisce l'occupazione di memoria e rende più veloce il programma, in quanto non deve ogni volta ricercare i valori utilizzati.
Dopo questa parte inizia il vero e proprio algoritmo, che comincia con la funzione principale (obbligatoria) *int main*.
Si va a dichiarare una variabile di tipo file, denominata fs, sia in scrittura che lettura utilizzando *fstream*. Si inseriscono anche le variabili c (di tipo char), v e w (di tipo int) alle quali si assegna il valore 0.
Si va ad assegnare un nome al file, e per completezza se ne inserisce anche la destinazione all'interno del pc. Fatto questo lo si va ad aprire e lo si imposta in modalità scrittura.
Tramite il comando *cout* (che mi permette di scrivere nell'output standard, ossia lo screen) si vanno ad inserire due righe che dicono il nome del file a cui si sta accedendo ed il fatto che per uscire bisogna premere il tasto esc.

Successivamente si è inserito un ciclo while, che va a eseguire dei blocchi d'istruzione a seconda dei casi indicati, che nel codice corrispondono alle frecce direzionali e dalla barra spaziatrice.
Col comando default si indica che se non si verifica nessuna delle istruzioni descritte sopra il codice esegue tale blocco.
Se si preme il tasto esc il programma esce dal ciclo.
Prima di analizzare i vari casi si va a dire che la variabile c è quella da utilizzare per l'immissione di dati da tastiera e si utilizza la funzione *getch*, che legge l'immissione da console del carattere senza passare per il buffer (altrimenti mentre si attende l'immissione di dati da tastiera questo occupa temporaneamente memoria per verificare che effettivamente si sia premuto un qualche tasto) a senza stampare a schermo il carattere utilizzato.
Andiamo ora a descrivere la varie situazioni che si possono verificare:

- tasto esc: esce dal ciclo;
- tasto freccia su: la dicitura *v++* va ad indicare l'aumento di 1 della variabile v; successivamente se il valore di v supera o eguaglia 10, il valore che si avrà sarà comunque 10 (questo limite è stato inserito perché si vuole limitare la velocità);
- tasto freccia giù: la dicitura *v--* va ad indicare la diminuzione di 1 della variabile v; anche in questo caso si fa in modo da limitare la velocità a -10;
- tasto freccia destra: la dicitura *w++* va ad indicare l'aumento di 1 della variabile w; al momento per questa variabile non era stato posto nessun limite;
- tasto freccia sinistra: la dicitura *w--* va ad indicare la diminuzione di 1 della variabile w; anche qui non è stato posto nessun limite;
- tasto barra spaziatrice: si è considerato l'inserimento di un tasto d'emergenza che andasse a porre entrambe le variabili uguali a 0;
- altri tasti: nessun effetto.

Dopo aver digitato un tasto, in ogni situazione si è inserito il comando *cout* per visualizzare sul terminale il valore che assume la variabile. Nei due casi riguardanti la velocità lineare v si è inoltre moltiplicato il valore per 0.1, così da andare ad ottenere un range di velocità compreso tra 1 e -1 m/s (si è fatto così perché per semplicità la variabile è stata definita come numero intero e quindi non accetta decimali).

Una volta premuto esc, si esce dal ciclo while e, prima della chiusura del file, si è deciso di inserire una dicitura che va a riassumere il valore assunto dalle due variabili alla fine delle varie operazioni svolte.
Successivamente l'ultima operazione è la chiusura (col salvataggio del file).

In base alla modalità di utilizzo del file, si ha che ogni volta che esso viene riaperto viene sovrascritto dalle nuove operazioni. Per evitare la sovrascrittura del file si può utilizzare la modalità append, che va ad aggiungere le nuove operazioni in coda a quelle precedenti.

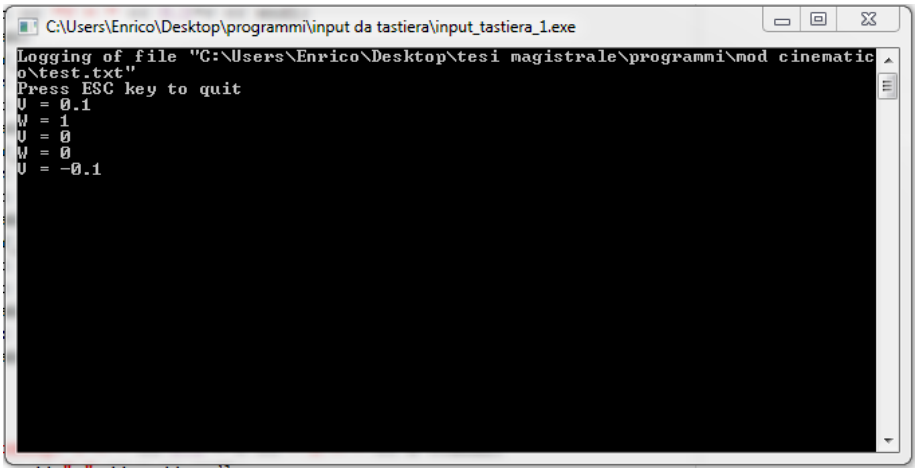

Fig 4.4 Esempio dal terminale con alcuni input da tastiera.

Andiamo ora a descrivere il secondo programma, che va a prendere i dati di output del programma precedente e a rielaborarli per ottenere le velocità angolari dei gruppi di ruote destro e sinistro.
Anche qui la prima operazione effettuata è stata quella di inserire le direttive *#include*, che in questo caso sono:

- *#include <stdio.h>* : contiene le definizioni di macro, costanti e dichiarazioni di funzioni e tipi usati per le varie operazioni di input/output;
- *#include <stdlib.h>* : qui vengono dichiarate funzioni e costanti di utilità generale, tra cui allocazione di memoria, controllo dei processi e altre funzioni comprendenti anche i tipi di dato.

Successivamente di sono dichiarate le due costanti R e C, la prima rappresentante il raggio delle ruote e la seconda rappresentante la lunghezza della carreggiata (ossia la distanza tra i centri delle due ruote sullo stesso asse).
Dopo si è riportata la funzione principale *main*, e si sono dichiarate le variabili di tipo double *wsx* e *wdx*.
La parte successiva del codice ha la funzione di warning: infatti se non si trova il file nella destinazione indicata oppure i dati che esso contiene sono più di due, il codice restituisce un messaggio di errore.

Poi esso stampa sul terminale i valori di C,R, V e W e, successivamente, usando le formule della cinematica già trovate nel capitolo 3, va a determinare le wdx e wsx.

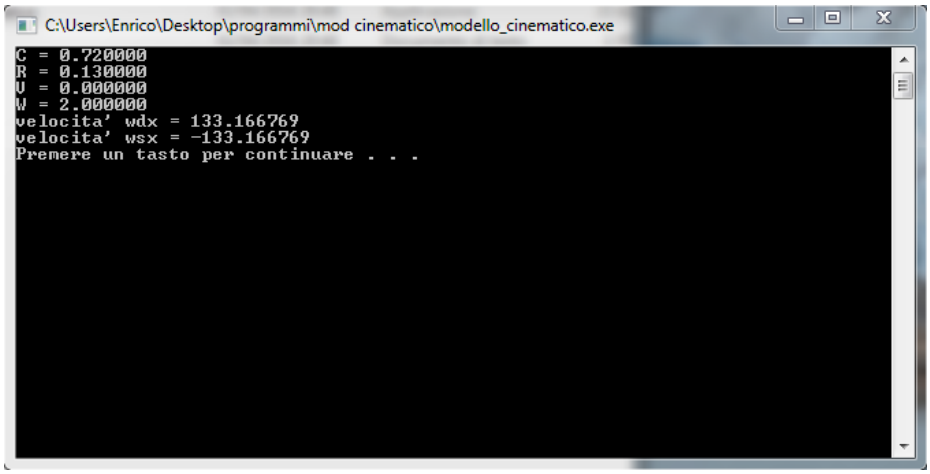

Fig. 4.5 L'output a video con le velocità della ruote calcolate.

4.3.2 Il codice unico

Successivamente al sistema di controllo si sono aggiunti i microntrollori e ciò ha portato alla necessità di dover avere uscite digitali dalla scheda UDOO.
Si è andati così a riscrivere un solo codice che girerà esclusivamente nel processore principale della scheda, così da aver disponibili le uscite USB necessarie al collegamento con i microcontrollori.
Tale codice è stato sviluppato interamente in Linux, così che potesse subito esser provato nella scheda; ciò ha portato a delle complicazioni impreviste, che verranno analizzate subito dopo aver descritto il programma.
Essenzialmente questo codice attende degli input da tastiera e da in uscita le velocità angolari dei gruppi di ruote destra e sinistra, i quali vengono inoltre salvati su un file.txt.
Inoltre si va a salvare un altro file, contenente l'elenco di tutti i comandi di input trasmessi, in maniera tale da poter successivamente ricostruire la traiettoria del rover.

Il programma inizia andando ad inserire le costanti, tramite la direttiva *#define*, le quali sono:

- *#define RAGGIO* : necessario per andare ad indicare il valore del raggio delle ruote;
- *#define ASSE* : con questa direttiva si va ad inserire il valore della distanza tra i centri delle due ruote sullo stesso asse;
- *#define CONVERSIONE* : mediante questa costante si va a riportare il valore della variabile W in maniera tale che la velocità tangenziale massima della ruota sia pari a 1 m/s.
  Tale argomento verrà trattato meglio più avanti.

Successivamente si sono andate a scrivere le varie direttive *#include*, le quali sono necessarie per poter sfruttare le librerie del compilatore e rende più semplice la scrittura del codice.
Essendo cambiato il sistema operativo, sono cambiate anche le librerie e quindi anche le direttive saranno differenti.
Le direttive che sono state usate sono:

- *#include <iostream>* : richiama dalla libreria standard di C++ la funzione iostream, necessaria se voglio leggere/scrivere qualcosa dallo stream (flusso);

- *#include <fstream>* : necessaria se voglio andare ad effettuare operazioni di input/output da file, in particolare con questa è possibile eseguire sia lettura che scrittura da file;
- *#include <stdlib.h>* : è l'header file che, all'interno della libreria standard del C++, dichiara funzioni e costanti di utilità generale come l'allocazione della memoria, controllo dei processi, e altre funzioni generali comprendenti anche i tipi di dato;
- *#include <ncurses.h>* : i compilatori impostati per sistemi operativi non DOS (Disk Operating System), come Linux Ubuntu, utilizzano la libreria curses al posto di conio.h. questo fatto ha portato a diverse problematiche durante la scrittura e compilazione del codice, come vedremo dopo. Per poter utilizzare questa libreria bisogna linkarla dalla shell del compilatore, infatti bisogna "informare" il compilatore che la si sta utilizzando; per fare ciò bisognava andare a digitare *gcc -o nomeprogramma -Wall "nomeprogramma".cpp –lncurses*; questo indica che ogni qualvolta si fosse eseguito il programma si sarebbe dovuto riportare riportare tale dicitura.

  Questo inconveniente è stato risolto andando, in CodeBlocks, a creare un nuovo progetto e ,all'interno di questo, si è andati ad aggiungere tra i collegamenti alle librerie standard anche quello alla libreria ncurses.h;
- *#include <math.h>* : questa libreria include tutta una gamma di funzioni utili per effettuare comuni trasformazioni ed operazioni matematiche.

Prima dell'algoritmo si è inserito il comando *using namespace std*, ciò viene fatto per andare a richiamare tutta la classe di funzioni standard (ad esempio cin e cout vengono richiamate da qui, così non bisogna specificare ogni volta che vengono chiamate).

Successivamente si sono andate a scrivere due funzioni, chiamate *compute_wspeed_dx* e *compute_wspeed_sx*, le quali servono per poter determinare le velocità angolari (in rad/s) dei gruppi di ruote destra e sinistra.
Esse sono definite come *float*, in modo da poter avere numeri con delle cifre decimali, a partire da valori di input di tipo int, ossia interi, che sono le due velocità (lineare e di rotazione del rover) in uscita dall'algoritmo principale.
Analizzando il caso della velocità angolare destra la formula per il suo calcolo è la seguente:

$$\text{wspeed\_dx} = (1 / \text{RAGGIO}) * ((\text{speed} * 0.1) + (\text{ASSE} * \text{rot}_{\text{speed}} * \text{CONVERSIONE}))$$

Questa è lievemente diversa dalla formula del modello cinematico in quanto:

- La velocità lineare *speed* viene moltiplicata per un termine 0.1; infatti per avere un codice più semplice si sono usati incrementi ai valori delle variabili pari a 1, e limitando i valori massimi e minimi a +10 e -10 (che diventano 1m/s e -1m/s, limiti imposti per la velocità di avanzamento del rover); comunque ciò verrà spiegato meglio nella parte di algoritmo in cui si definiscono gli imput da tastiera;
- La velocità di rotazione del rover *rot_speed* viene moltiplicata per un fattore di conversione; il fatto è che anche questa variabile, data come input da tastiera, assume valori tra +10 e -10 ma non rappresenta realmente la velocità di rotazione, ma indica il fatto che tra la velocità nulla e quella massima si possono andare ad imporre 10 valori.
  Il ragionamento effettuato per calcolare il valore del fattore di conversione è il seguente:
  si è voluta attribuire anche alla rotazione del rover una velocità periferica della ruota pari ad 1 m/s in condizione di pura rotazione.
  Si è ottenuta allora la seguente espressione:

$$v_{tan} = 1 = AXIS * rot_{speed} * CONVERSION$$

  Dove rot_speed è stata posta pari a 10 (valore massimo) e la costante AXIS è nota e pari alla distanza dei punti di contatto delle ruote sulle stesso asse.
  Si ottiene infine il valore 0.1338 rad/s.

Una volta determinato il valore della velocità della ruota, si è andati a moltiplicarlo per 1000 e successivamente a usare la funzione *trunc* per troncare le cifre dopo la virgola. Infine si è diviso il valore ottenuto per 1000.
Ciò è stato fatto per poter avere tutti i dati con tre cifre decimali.

Lo stesso procedimento si è applicato alla funzione per il calcolo della velocità del gruppo sinistro di ruote, con ovviamente la giusta formula del modello cinematico.

Dopo la definizione delle due funzioni, si è andati a scrivere l'algoritmo vero e proprio.
Questo inizia sempre con la funzione principale *int main*, dove int è un identificatore che indica che la funzione main (presente nella libreria standard del C++) restituisce un valore intero, pari a zero se il programma è stato eseguito correttamente. Il fatto che restituisce tale valore è assicuarto dal fatto che in fondo alla funzione si va a scrivere *return 0*.
La funzione main rappresenta il corpo del programma principale.

Successiavemente si vanno ad aprire due stream con due diversi nomi, uso detto *fs_linear* e l'altro *fs_wheel*. Si sono inoltre creati i nomi dei file che verranno associati agli stream definiti sopra, i quali sono rispettivamente:

- filename_linear, chiamato o_linear.txt; quando si va ad aprire lo stream fs_linear gli si va ad associare questo filename;
- filename_wheel, chiamato o_wheel.txt; quando si va ad aprire lo stream fs_wheel gli si va ad associare questo filename.

Si sono anche dichiarate le variabili desiderate, in questo caso le variabili consistono in *speed* e *rot_speed*, entrambe di tipo int (numeri interi) ed impostate con valore iniziale nullo, oltre che nella variabile ch che servirà per gli input da tastiera.
Si è poi inizializzata la modalità curses andando a scrivare *initscr()*:allo scopo di usare il pacchetto screen, le routine devono essere informate delle caratteristiche del terminale, e bisogna allocare spazio per curscr e stdscr. La funzione initscr() fa entrambe queste cose.
Successivamente si è disabilitato il buffering dell'output inserendo il commando *raw()*: questo è necessario in quanto in Linux non è presente la libreria conio.h la quale in automatico lo disabilita.
Il comando *keypad(stdscr, TRUE)* ha permesso l'accesso a comandi della tastiera che in Linux altrimenti non sarebbero stati disponibili (ciò verrà spiegato dopo), come i tasti funzione, le frecce direzionali,ecc..
Con la routine *noecho()* si è fatto in modo che i caratteri corrispondenti ai tasti premuti non apparissero sullo schermo; in questo modo nello screen si avranno solo la variabili di interesse.

Poi si è impostata la variabile ch in modo che corrispondesse ai comandi da tastiera; ciò è stato fatto utilizzando la funzione *getch()*. Questo normalmente in Linux avrebbe portato a dei problemi, in quanto qui con getch ogni volta che si preme un tasto bisogna anche premere invio, altrimenti il comando non viene elaborato. Logicamente questa è un'azione poco pratica ai fini del controllo del moto di un rover, e si è dovuta trovare una soluzione.
La funzione che ha permesso di risolvere questo problema è ncurses, però il suo utilizzo ha portato ad altri inconvenienti che, nel paragrafo dedicato, verranno analizzati.
Poi si è andato ad impostare un ciclo while che andasse ad eseguire dei blocchi d'istruzione secondo i casi indicati.

Andiamo ad analizzare i vari casi:

- Caso KEY_UP : premendo la freccia direzionale up, si ottiene l'incremento di un'unità della variabile speed, che andrà poi stampata sul terminale. Per semplicità tale variabile è stata impostata di tipo int, ed inoltre si è posto come limite superiore il valore 10. Successivamente si è descritto un ciclo if che mi riportasse tali valori nel range effettivamente desiderato per le velocità, ossia tra +1 e -1 e con passo 0.1. Ciò si è ottenuto aggiungendo al ciclo if un'istruzione che, se la variabile fosse uguale o maggiore di 10, a schermo si andasse a stampare il valore 1, mentre se era minore di 10, si andasse a stampare il valore della variabile, ma preceduto da 0. Questo ha però portato il problema che, se la variabile speed era negativa, a schermo si sarebbe osservata la scrittura 0.- ; questo bug è stato risolto andando s a scrivere i valori assoluti della variabile, tramite la funzione *abs*, aggiungendo un'istruzione che, nel caso in cui la velocità sia negativa, a schermo si va a stampare il risultato preceduto dal segno meno;
- Caso KEY_DOWN : analogo dal caso key_up quando si va a premere la freccia direzionale down. Qui si ottiene la diminuzione del valore della variabile speed di un'unità. Si è posto il limite inferiore pari a -10 e si è usata la stessa procedura del caso sopra per ottenere come uscita sul terminale i valori compresi tra -1 e +1;
- Caso KEY_LEFT : premendo la freccia direzionale sinistra, si va ad ottenere la diminuzione della variabile rot_speed di un'unità. Questo caso è simile a quello descritto della freccia

direzionale down, solo che interessa la variabile rot_speed invece di speed, e sul terminale va a stampare il valore W invece che V. Per questa variabile inoltre non si è andati a limitarla tra -1 e +1 con passo 0.1, ma si è mantenuto il passo unitario e si sono impostati i limiti tra -10 e +10;

- Caso KEY_RIGHT : premendo la freccia direzionale destra si ottiene l'incremento di un'unità della variabile rot_speed; è molto simile al caso con la freccia direzionale sinistra;
- Caso BACK_SPACE : si è impostato questo comando come un tasto di emergenza, infatti se si desidera annullare tutte le velocità, con la pressione della barra spaziatrice si ottiene la stampa su terminale di entrambe le velocità con valore nullo;
- Caso KEY_(F2) : in Linux il tasto esc non funziona per ottenere l'uscita dal terminale, inoltre il tasto F1 è impostato come collegamento rapido ad una guida. Per evitare ogni tipo di problema si è impostato il tasto funzione F2 come tasto di uscita dal terminale.

Infine si è andati a scrivere, nello stream fs_linear, i valori di V e W, ossia delle velocità lineare e di rotazione del rover. La velocità lineare è stata moltiplicata per 0.1 così da ottenere valori compresi tra -1 e 1, mentre la velocità di rotazione è stata moltiplicata per la costante CONVERSIONE, che la porta in rad/s e con valori effettivi e non relativi.
L'inserimento della scrittura *\t\t* tra i due risultati che voglio salvare sul file va semplicemente a porre due spaziature tra le due variabili, in maniera da ottenere due colonne allineate di valori.
Poi si è andati a scrivere, nello stream fs_wheel, i valori delle velocità angolari delle ruote, ottenute tramite le funzioni che applicano il modello cinematico dati gli input di velocità.
Come ultima operazione, si è disattivata la modalità curses con il comando *endwin()*.

4.3.3 Problematiche incontrate con Linux Ubuntu

La scrittura del programma utilizzando come sistema operativo Linux Ubuntu, invece di Windows, ha portato a diversi problemi e complicazioni, che essenzialmente si possono ricondurre a due ragioni:

- differenze basilari a livello di libreria del compilatore;
- difficoltà nel far si che il programma leggesse gli input da tastiera, in particolare le frecce direzionali.

La più grossa problematica riscontrata durante la scrittura dell'algoritmo è stata l'impossibilità di poter utilizzare la libreria <conio.h>, definita solo in Windows in quanto libreria non standard.
In Windows la libreria conio.h permette l'utilizzo della funzione getch, la quale viene utilizzata per ottenere dei dati medianti inserimento da tastiera.
In Linux, come prima soluzione, si era provata la funzione getchar (definita senza nessuna problematica in Linux), la quale però è lievemente differente da getch. Infatti, mente getch da un ritorno immediato a video del tasto scelto, per getchar bisogna dare un tasto EOL (end of line).
Il fatto di dover premere un altro comando per far leggere il valore di input da tastiera si è rivelato molto scomodo, in quanto poco pratico per gli scopi del programma.
Allora si è deciso di utilizzare la libreria ncurses, la quale ha subito presensato un problema.
Infatti all'interno del compilatore CodeBlocks, per poter far girare il programma, si è dovuto aprire un nuovo progetto e inserirci, oltre ai collegamenti di default con le librerie standard, anche un collegamento alle libreria <ncurses.h>.
Successivamente si è utilizzata la funzione getch, anche se lavora in maniera differente rispetto a Windows.
Infatti questa lavora in modalità buffer come default, ciò implica due cose:

- Se l'input viene gestito mediante un buffer, tutti i valori (ossia i codici ASCII) dei caratteri inseriti da tastiera vengono salvati temporaneamente nel buffer e rimangono li finchè l'utente non preme il tasto ENTER. A questo punto, dal buffer vengono prelevati tanti caratteri quanti la funzione richiamata all'interno del programma ne può accettare; ad esempio, se uso una funzione che prevede l'immissione di un solo carattere, dal buffer viene prelevato solo il primo carattere. Gli altri rimangono disponibili per successive chiamate alla funzione;

- Il terminale continua a portare a schermo 0 finchè non si preme un tasto; questo in pratica è dovuto al fatto che il codice continua a verificare che effettivamente non si sia premuto un tasto. In Windows questa problematica è stata eliminata dalla libreria conio.h

Per risolvere questi due inconvenienti si è dovuta quindi attivare le curses mode, col comando *initscr()*, e poi andare a disattivare il buffering tramite il comando raw(), il quale va a disattivare il buffer tramite l'apertura di un'altra sessione di terminale (che praticamente rende quella bufferizzata non visibile).
Inolter per eliminare il fatto che si visualizzasse il tasto premuto si è inserito il comando *noecho()*.

Infine, con la modalità curses attiva si è riscontrato il problema del non funzionamento degli oggetti *cin* e *cout*, per cui si è andato ad inserire il comando *printw()*.
Questo è l'equivalente in ncurses della funzione printf(), la quale però presenta degli svantaggi rispetto a cin e cout.
Prima di tutto, cin e cout sono type-safe. Con printf si è obbligati a dichiarare il tipo della variabile da stampare (nel nostro caso "%d", che indica che sul terminale si stamperà un valore di tipo int) e poi il nome della variabile.
Se il tipo della variabile non coincide con quello dichiarato, il compilatore non risponde con un errore ma il programma avrà un comportamento non prevedibile. Con cout, invece, il tipo della variabile viene dedotto automaticamente dalla variabile stessa.
Inoltre cin e cout sono in realtà stream, e quindi possono essere trattati come generici flussi di dati. Gli stream (flussi) sono oggetti che astraggono una semplice comunicazione:

- da una sorgente (produttore);
- ad una destinazione (consumatore).

Nel caso di cin la sorgente è la tastiera, mentre la destinazione è il programma stesso; in cout, invece, è il programma ad essere la sorgente, mentre la destinazione è il video.

Analizziamo ora la seconda fonte di problemi, ossia la difficoltà da parte del compilatore a leggere gli input da tastiera, ed in particolare leggere le frecce direzionali.

Prima della scrittura del vero e proprio codice, si è voluto che il programma mostrasse come uscite video il nome dei tasti premuti su tastiera.

Si è visto che, all'inserimento delle frecce direzionali, il programma dava in uscita una combinazione di altri tasti, non coincidenti con quanto voluto.

Questo è dovuto al fatto che, in Linux, il compilatore di default non legge i caratteri ASCII associati ai tasti funzione (e quindi anche alle frecce direzioniali) perché superiori ad un certo valore; così in uscita da una serie di valori che corrispondono ai caratteri ASCII di altri comandi.

La soluzione è stata trovata mediante l'inserimento del comando *keypad(stdscr, TRUE)*, il quale permette "l'accesso" ai tasti funzione e alle frecce direzionali.

Nell'immagine sotto si vede il problema che si genera senza tale comando: invece della lettura della freccia "su", il programma interpreta il codice corrispondente come la somma di diversi caratteri, che nell'esempio sono ^[[A .

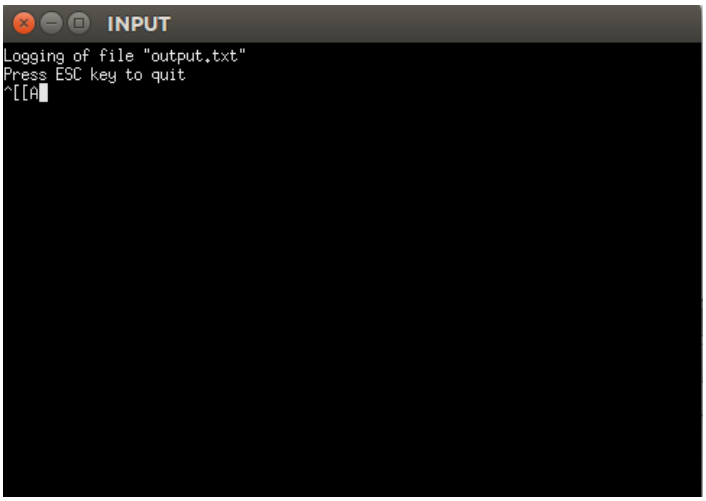

Fig. 4.6 Immagine delle uscite a video con la pressione del tasto freccia "su" se non si attivano i comandi funzione. Al posto di un singolo tasto, il programma interpreta il codice ASCII come una sequenza di più tasti che hanno insieme lo stesso codice delle frecce.

A titolo di esempio, andiamo a vedere il contenuto dei due file generati dal codice, in due situazioni:

- speed = 1 e rot_speed = 0, ossia velocità lineare positiva e rotazione del rover nulla;
  si ottengono i seguenti file:

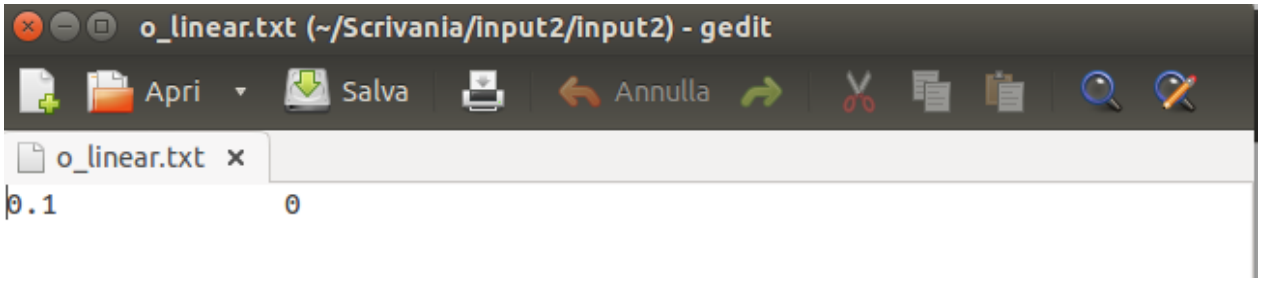

Fig. 4.7 File o_linear.txt nel quale si possono individuare le due uscite, ossia la velocità lineare e quella di rotazione del rover.

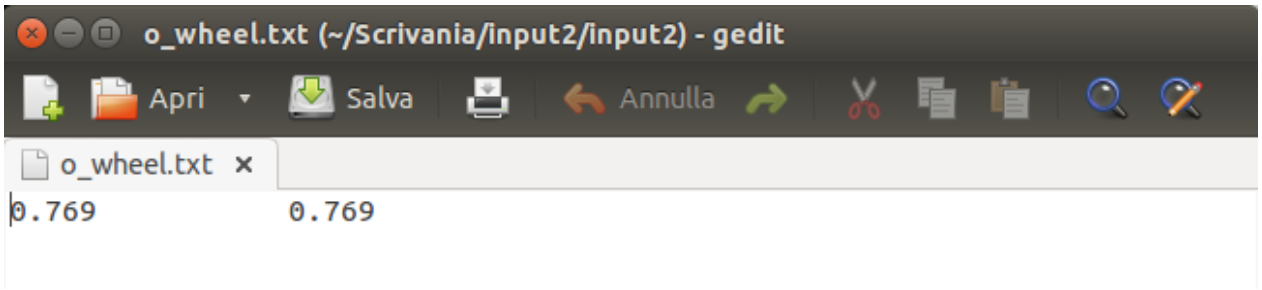

Fig. 4.8 File o_wheel.txt nelquale si possono vedere i valori delle velocità angolari in rad/s da applicare alle ruote sinistre e destre.

- speed = 0 e rot_speed = 1, ossia il rover ruota attorno a se stesso col verso scelto positivo; in questa situazione i file che si ottengono sono:

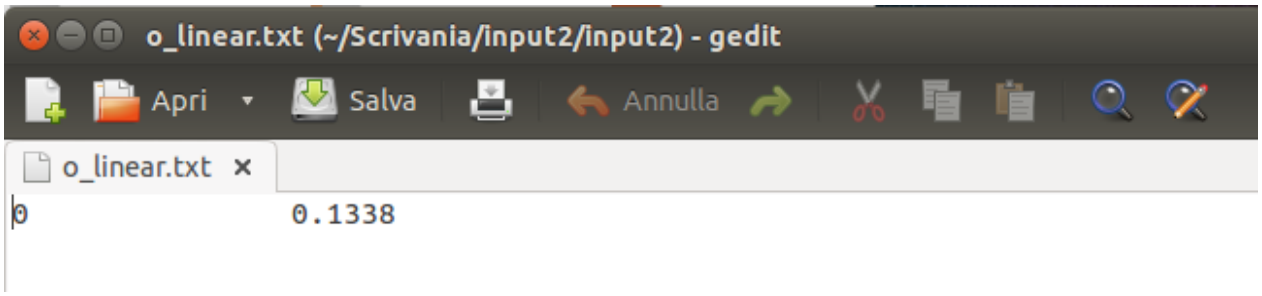

Fig. 4.9 File o_linear.txt in cui sono visibili le due uscite.

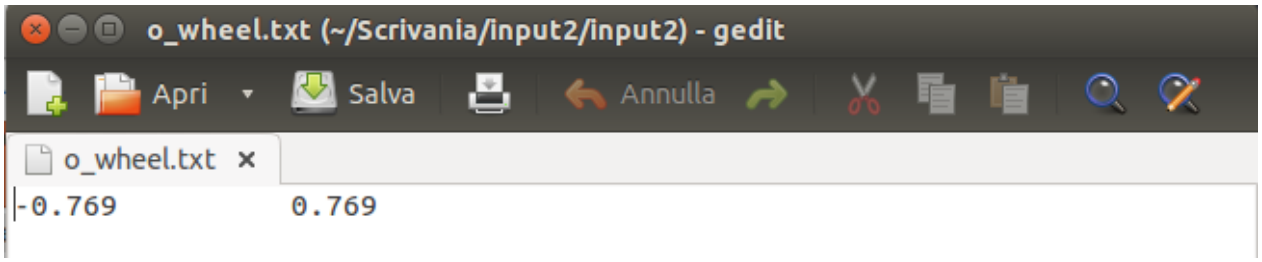

Figura 4.10 File o_wheel.txt in cui sono presenti le velocità angolari dei gruppi di ruote.

CAPITOLO 5

Simulazioni di validazione

In questo capitolo si riportano alcuni test fatti per la validazione dell'algoritmo di controllo scritto.

Per fare queste validazioni si è sfruttato la strumento Matlab, che ha permesso di eleborare in forma grafica i vari dati di input ed output che sono contenuti nei file d'uscita dal codice scritto.

Per questa validazione si sono andati a considerare tre differenti casi:

- velocità esclusivamente nella direzione d'avanzamento del rover, non si è quindi andati a considerare la rotazione;
- pura rotazione del rover, in quanto si è mantenuto il robot fermo e lo si è fatto girare;
- velocità miste, dove si è considerato un input generico da parte dell'operatore.

Per fare questo si è considerato che ogni comando (e di conseguenza ogni input di velocità) avesse la durata di un secondo; si arrivano ad ottenere così dei grafici discreti, in quanto ogni input da tastiera rappresenta uno step tra diversi valori possibili.

Il codice Matlab utilizzato può esser trovato in appendice.

5.1 1° caso: velocità puramente lineare

In questo primo caso si è andati ad ipotizzare esclusivamente velocità lineari per il rover, quindi esso si muove in linea retta senza nessuna rotazione.

Si nota dai grafici che le velocità di input e quelle di output costituiscono delle funzioni discrete: ciò significa che all'interno del tempo in cui vale quel range di velocità (nei casi in esame posto pari ad un secondo) si avrà che il valore non cambia.

Si nota inoltre che, essendo una velocità lineare senza nessuna rotazione, le velocità angolari dei gruppi di ruote destro e sinistro saranno uguali, e ciò conferma il modello utilizzato per la scrittura dell'algoritmo.

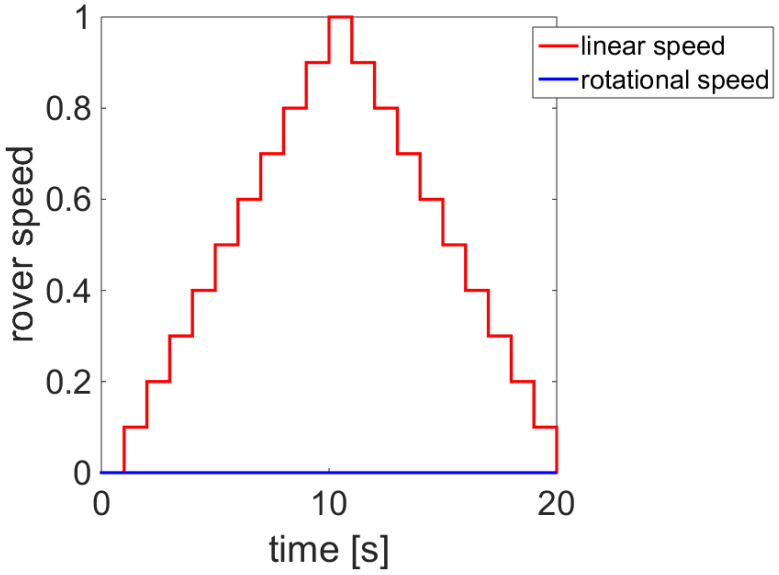

Fig. 5.1 Grafico delle velocità di input, ossia quelle scelte dall'operatore.

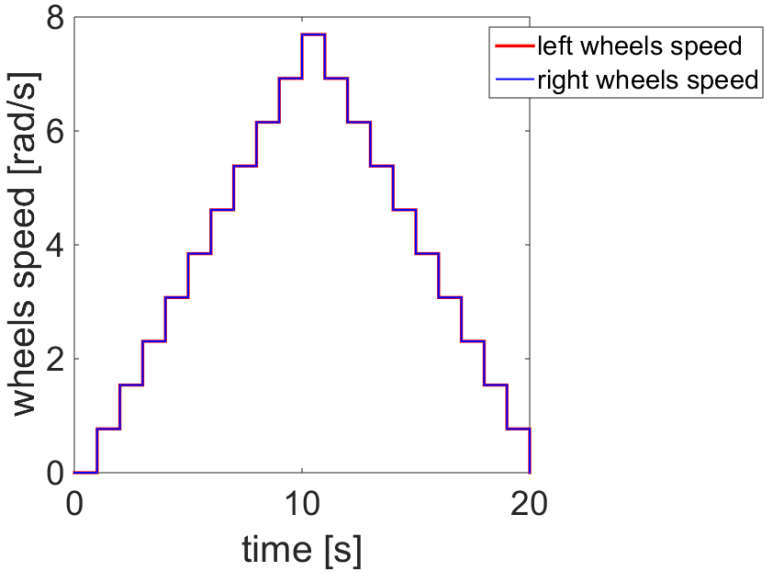

Fig. 5.2 Grafico delle velocità angolari dei gruppi di ruote; come atteso sono uguali.

Dalle immagini si nota il fatto che sono funzioni discrete, in quanto vengono rappresentate con un grafico a forma di gradino.

5.2 2° caso: pura rotazione del rover

Nella seconda analisi di validazione, si è andati a considerare una pura rotazione del rover, per cui si avrà che la velocità lineare è sempre nulla.

Dai grafici si nota che per ottenere una pure rotazione del mezzo, le velocità angolari dei gruppi di ruote saranno in modulo uguali ma con segno opposto.

Ciò è giustificato dal fatto che per ottenere una pura rotazione i due gruppi di ruote devono essere controrotanti.

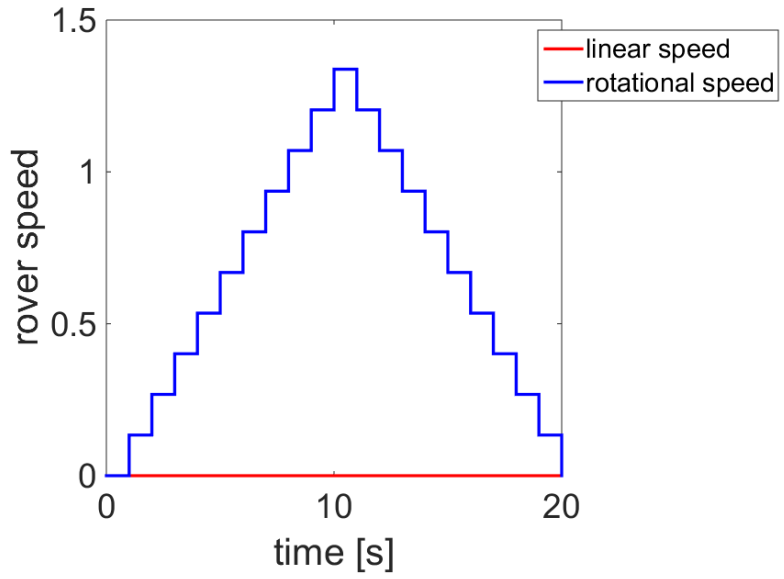

Fig. 5.3 Grafico delle velocità in input; si nota che la velocità lineare è sempre nulla, così da avere una pura rotazione del rover.

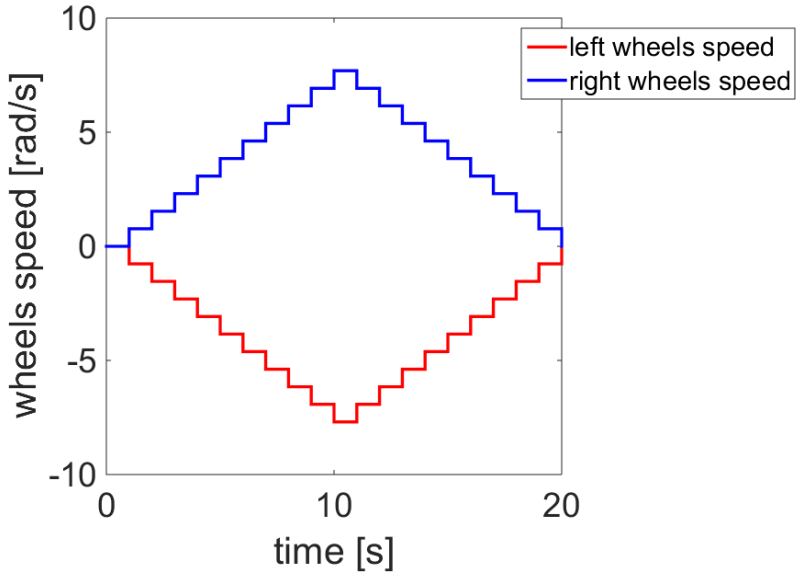

Fig. 5.4 Grafico delle velocità angolari in uscita, si nota che sono uguali in modulo ma con segno opposto.

5.3 3° caso : input casuale di velocità

Come ultima analisi di verifica del codice, si sono considerati degli input casuali di velocità.

Qui si avrè sia rotazione che traslazione del rover e, quindi, per sterzare non si avranno più velocità uguali in modulo ma con segno opposto; accadrà che una velocità sarà semplicemente maggiore dell'altra per poter così ottenere la rotazione del robot nella direzione desiderata.

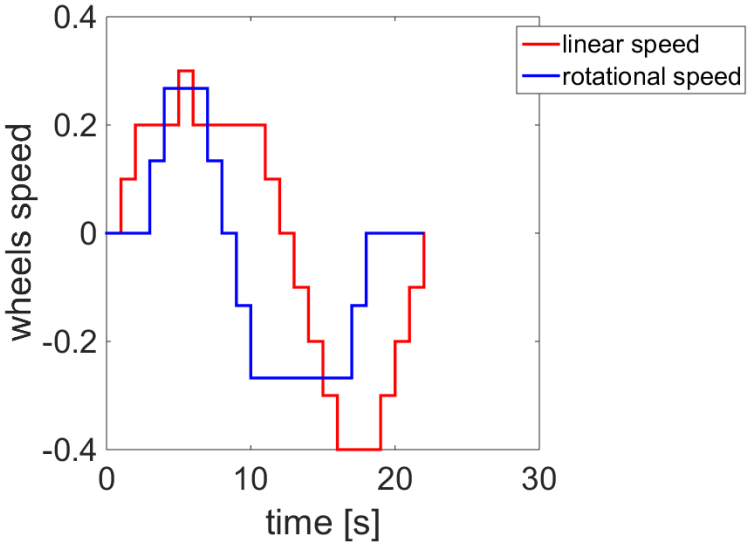

Fig. 5.5 Grafico degli input casuali di velocità.

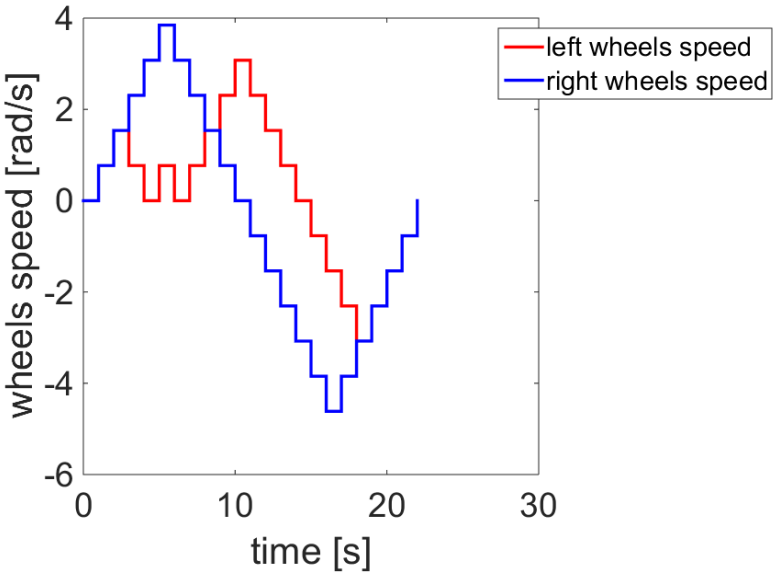

Fig. 5.6 Grafico delle velocità angolari in uscita.

Conclusioni

Morpheus, nato dall'omonimo progetto di un gruppo di studenti dell'Università di Padova, è un prototipo di rover marziano che a settembre 2016 parteciperà alla ERC (European Rover Challenge). La sua locomozione è assicurata da sei ruote fisse non sterzanti disposte in configurazione skid-steer, semplice e nello stesso tempo robusta.

Le ruote sono connesse al telaio a gruppi di due mediante dei bilancieri, i quali effettuano il collegamento mediante perni montati su cuscinetti. Due di questi bilancieri sono montati a fianco del rover, mentre il terzo (quello posteriore) è installato in modo da poter compensare l'angolo di roll. Tale configurazione permette di non utilizzare sospensioni, in quanto in ogni caso l'aderenza al terreno è garantita per le sei ruote.

Ognuna di queste ruote è connessa ad un motore brushless; questi, che incorporano un driver, sono collegati a dei microcontrollori che li mettono in comunicazione con la scheda elettronica che si occupa del controllo del moto.

Come scheda la scelta è ricaduta sulla UDOO Quad, per la sue performance in relazione al costo e per la possibilità di poter disporre sia di uscite analogiche che digitali.

Il primo passo di questo progetto di tesi è stato l'installazione del sistema operativo nella scheda, avvenuto mediante l'utilizzo di una microSD. Come sistema si è scelto di utilizzare l'OS UDOObuntu, di base Linux. Durante questa fase si sono provate anche varie configurazioni per la scheda per quanto riguarda il collegamento con uno schermo. Tra la varie prove, si è vista la possibilità, oltre che di connessione diretta con uno schermo, di poter collegare la scheda tramite rete wifi col pc dell'operatore, e successivamente di poter sfruttare programmi di condivisione schermo (si è usato VNC screen) per poter usare il monitor del computer al posto di uno esterno (ciò è stato fatto perchè ovviamente nel rover uno schermo non sarebbe di nessuna utilità, quindi si è cercato un metodo per connettersi in remoto alla scheda).

Inoltre per poter connettere la scheda via USB al computer (necessario per trasferire file) si è anche andati ad effettuare il debugging USB proprio per poter connettere la scheda via porta seriale e per capire come poter risolvere problemi legati al software installato.

Una volta terminata la fase di preparazione della scheda elettronica, si è passati alla fase della scrittura del sistema di controllo del moto del rover, da inserire nella scheda.
Si è andati s scrivere la parte di controllo di alto livello, ossia quella che va a decidere che velocità imporre al robot. Per il completamento della restante parte del controllo bisognerà attendere che siano disponibili i microcontrollori ed i motori, per collegare la velocità voluta a un input leggibile dai motori.
Per ottenere questo si è andati a studiare un modello cinematico e si è scelto di usarne uno relativamente semplice, valido per poter eseguire la prima parte dei test.
Questo va a considerare il rover con una configurazione differential drive, ossia va a dividere la velocità delle ruote in due gruppi, destro e sinistro, con i quali si ottiene la movimentazione del rover. Grazie al fatto che i motori sono controllati in velocità è sufficiente l'utilizzo del modello cinematico, quello dinamico potrà essere introdotto una volta iniziata la campagna di test sul rover, così da poter ottenere alcuni parametri sperimentali necessari per modellarlo.
Ovviamente l'utilizzo della configurazione differential drive porterà a degli errori, ma come strumento iniziale rappresenta una buona approssimazione.
Tale modello è stato inserito all'interno di un programma che, dati degli input da tastiera (tramite le frecce direzionali), va prima ad indentificare le velocità lineare e di rotazione del rover, poi , tramite le formule ricavate dalla cinematica, le convertirà in velocità angolari delle ruote, una per il gruppo di destra ed una per il gruppo di sinistra.
L'utilizzo del sistema operativo Ubuntu ha portato a varie problematiche riguardo la scrittura del codice, in quanto molte delle librerie di Windows non sono presenti, e ciò ha portato ad una lunga ricerca per determinare le giuste funzioni da utilizzare.
Tale codice, comunque, resterà valido anche nello sviluppo futuro del modello, in quanto è solamente necessario modificare le funzioni definite per inserire il modello cinematico del rover. Inoltre potrebbe essere possibile la sua implementazione in un joystick, che andrebbe a migliorare il moto del rover.
Lo sviluppo futuro più interessante è comunque la fase dei test, che verrà intrapresa non appena saranno disponibili i microcontrollori ed i motori.
Con questa sarà possibile andare a studiare l'efficienza del sistema di controllo definito ed, inoltre, si potrà andare a definire un modello cinematico più complesso,basato su dati sperimentali.

Integrando questo lavoro con quello svolto da altri componenti del team si potrà anche andare a impostare la modalità di navigazione autonoma del rover, che è uno degli obiettivi che la competizione richiede di raggiungere.

Appendice

Qui si sono riportati i codici che sono stati scritti al termine della programmazione ed il codice Matlab usato per i test di validazione.

Si inizia andando a riportare i due codici scritti nella prima fase, quella in cui si desiderava avere i codici separati per poter sfruttare le uscite analogiche.

Si riporta ora il programma *keyboard_input*, il quale dati gli input da tastiera va a determinare le velocità lineare e di rotazione del rover, salvandole su un file:

```
#include <conio.h>
#include <iostream>
#include <fstream>

using namespace std;

#define KEY_ESC 27
#define KEY_UP 72
#define KEY_DOWN 80
#define KEY_LEFT 75
#define KEY_RIGHT 77
#define KEY_BACKSPACE 8


int main(int argc, char * argv[])
{
	fstream fs;
	char c = 0;
	int v = 0;
```

```
 int w = 0;
 const char * filename = "C:\\Users\\Enrico\\Desktop\\test.txt";

if (argc > 1)
   filename = argv[1];

fs.open (filename, fstream::out);

cout << "Logging of file \"" << filename << "\"" << endl;
cout << "Press ESC key to quit" << endl;

while(c != KEY_ESC) {

    c = getch();

    switch(c) {
    case KEY_UP:
        v++;
        if (v >= 10)
        {(v = 10);
        } else {v = v;
        }
        cout << "V = " << 0.1*v << endl;
        break;
    case KEY_DOWN:
        v--;
        if (v < -10)
        {(v = -10);
        } else {v = v;
        }
        cout << "V = " << 0.1*v << endl;
        break;
    case KEY_LEFT:
        w--;
```

```
            cout << "W = " << w << endl;
            break;
        case KEY_RIGHT:
            w++;
            cout << "W = " << w << endl;
            break;
        case KEY_BACKSPACE:
            cout << "W = 0 " << endl;
            cout << "V = 0 " << endl;
            break;
        default:
            break;
        }
    }

     cout << "Closing: V = " << 0.1 * v << " W = " << w <<endl;
     fs << 0.1 * v << " " << w << endl;
     fs.close();
     getch();

    return 0;
}
```

Andiamo ora a riportare la seconda parte del programma, quella che, presi i valori di velocità dal file creato alla file del codice precedente, va a determinare la velocità delle ruote del rover:

```
#include <stdio.h>
#include <stdlib.h>

static const double C = 0.7475;
static const double R = 0.13;
static const double Z = 0.1338;

int main(int argc, char * argv[])
{
      const char * filename = "output.txt";
    double wdx,wsx;
    double v,w;
      FILE * fd;
      int ret;

    if (argc > 1)
            filename = argv[1];


    fd = fopen(filename, "r");
      if(!fd) {
            perror("Errore in apertura del file");
            exit(1);
      }
    ret = fscanf(fd, "%lf %lf", &v, &w);
      fclose(fd);

    if (ret != 2) {
       puts("Unable to read input values");
       exit(1);
    }
```

```
      printf("C = %lf\n", C);
      printf("R = %lf\n", R);
      printf("V = %lf\n", v);
      printf("W = %lf\n", w);

    wdx = (1/R)*(v + C*w*Z);
    printf("velocita' wdx = %lf\n", wdx);
    wsx = (1/R)*(v - C*w*Z);
    printf("velocita' wsx = %lf\n", wsx);

    system("pause");

      return 0;
}
```

Infine si va a riportare il codice installato sulla scheda UDOO, quello che racchiude in un programma unico tutto l'algoritmo finora trattato:

```
#define RADIUS  0.13 //[m]
#define AXIS    0.7475 //[m]
#define CONVERSION     0.1338 //[rad/s]

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <ncurses.h>
#include <math.h>


using namespace std;


float compute_wspeed_dx(int speed, int rot_speed)
{
    float wspeed_dx;
    wspeed_dx =(1/RADIUS)*((speed*0.1)+(AXIS*rot_speed*CONVERSION));
    wspeed_dx = wspeed_dx * 1000;
    wspeed_dx = trunc(wspeed_dx);
    wspeed_dx = wspeed_dx / 1000;
    return wspeed_dx;
}

float compute_wspeed_sx(int speed, int rot_speed)
{
    float wspeed_sx;
    wspeed_sx =(1/RADIUS)*((speed*0.1)-(AXIS*rot_speed*CONVERSION));
    wspeed_sx = wspeed_sx * 1000;
    wspeed_sx = trunc(wspeed_sx);
    wspeed_sx = wspeed_sx / 1000;
```

```
    return wspeed_sx;
}


int main()
{
    int ch;
    ofstream fs_linear;
    ofstream fs_wheel;
    const char * filename_linear = "\o_linear.txt";
    const char * filename_wheel = "\o_wheel.txt";
    int speed = 0, rot_speed = 0;

    fs_linear.open (filename_linear);
    fs_wheel.open (filename_wheel);

    initscr();                      /* Start curses mode            */
    raw();                          /* Line buffering disabled      */
    keypad(stdscr, TRUE);           /* We get F1, F2 etc..          */
    noecho();                      /* Don't echo() while we do getch */

    ch = getch();
    while (ch != KEY_F(2))
    {
        switch(ch)
        {
            case(KEY_UP):
                speed++;
                printw("V =: ");
                if(speed<0) printw("-");
                if(speed>=10)
                {
                    speed=10;
                    printw("1.0");
```

```
        }
        else
        {
            printw("0.");
            printw("%d",abs(speed));
        }
        printw("  ");
        break;

    case(KEY_LEFT):
        rot_speed--;
        printw("W =: ");
        if(rot_speed<0) printw("-");
        if(rot_speed<=-10)
        {
            rot_speed=-10;
            printw("10");
        }
        else
        {
            printw("");
            printw("%d",abs(rot_speed));
        }
        printw("  ");
        //printw("FRECCIA SX\n");
        break;

    case(KEY_DOWN):
        speed--;
        printw("V =: ");
        if(speed<0) printw("-");
        if(speed<=-10)
        {
            speed=-10;
```

```
            printw("1.0");
        }
        else
        {
            printw("0.");
            printw("%d",abs(speed));
        }
        printw("  ");
        break;

    case(KEY_RIGHT):
        rot_speed++;
        printw("W =: ");
        if(rot_speed<0) printw("-");
        if(rot_speed>=10)
        {
            rot_speed=10;
            printw("10");
        }
        else
        {
            printw("");
            printw("%d",abs(rot_speed));
        }
        printw("  ");
        //printw("FRECCIA DX\n");
        break;
    case(' '):
        printw("V = 0 \n");
        printw("W = 0 \n");
        //printw("SPAZIO\n");
        break;
}
```

```
        fs_linear << 0.1 * speed << "\t\t" << CONVERSION * rot_speed <<
endl;
        fs_wheel << compute_wspeed_sx(speed,rot_speed) << "\t\t" <<
compute_wspeed_dx(speed,rot_speed) << endl;
        ch = getch();
    }
    fs_linear.close();
    fs_wheel.close();
    endwin();                       /* End curses mode  */

    return 0;
}
```

Andiamo, infine, ad elencare il significato dei colori usati:

- Verde : usato per i comandi #define e #include, ossia per definire costanti e richiamare le librerie;
- Rosso : usato per le parti di codice che si visualizzerano in output si video;
- Viole : usato per i numeri;
- Azzurro : utilizzato per le righe di commento.

Ora si va a riportare il codice matlab utilizzato per i test di validazione, nel quale si vanno a graficare vari test di velocità in input e di velocità angolari in uscita.
Ciò si è fatto per osservare a livello grafico la coerenz delle variabili in ingresso con quelle in uscita.

```
% codice che va a prendere i file .txt come input (quello che contiene
le velocità richieste)
% e lo porte in quelle angolari delle ruote, inoltre fa grafico
discreto,
% rispetto tempo (ogni comando ha timesteps di un secondo), di tutte e
4 le
% variabili


close all
clear all
clc

% carico i vari file

fname1 = 'C:\Users\Enrico\Desktop\tesi magistrale\materiale
matlab/linear1.txt';
fname2 = 'C:\Users\Enrico\Desktop\tesi magistrale\materiale
matlab/linear2.txt';
fname3 = 'C:\Users\Enrico\Desktop\tesi magistrale\materiale
matlab/linear3.txt';
fname4 = 'C:\Users\Enrico\Desktop\tesi magistrale\materiale
matlab/wheel1.txt';
fname5 = 'C:\Users\Enrico\Desktop\tesi magistrale\materiale
matlab/wheel2.txt';
fname6 = 'C:\Users\Enrico\Desktop\tesi magistrale\materiale
matlab/wheel3.txt';
```

```
input1 = load(fname1);
input2 = load(fname2);
input3 = load(fname3);
output1 = load(fname4);
output2 = load(fname5);
output3 = load(fname6);

% introduco ora la variabile time

time = [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20];

% definisco ora la matrice contenente gli ingressi della velocità
lineare
speed = zeros(21,2);

for j= 1:1:numel(input1(:,1))
    speed(j+1,1) = input1(j,1);
    speed(j+1,2) = input2(j,1);
end

% definisco ora la matrice contenente gli ingressi della velocità di
% rotazione del rover

rot_speed = zeros(21,2);

for j= 1:1:numel(input1(:,1))
    rot_speed(j+1,1) = input1(j,2);
    rot_speed(j+1,2) = input2(j,2);
end


% ora invece vado a definire la matrice contenente le uscite, cioè le
% velocità angolari delle ruote
```

```
w_wheel_sx = zeros(21,2);

for j= 1:1:numel(input1(:,1))
    w_wheel_sx(j+1,1) = output1(j,1);
    w_wheel_sx(j+1,2) = output2(j,1);
end

w_wheel_dx = zeros(21,2);

for j= 1:1:numel(input1(:,1))
    w_wheel_dx(j+1,1) = output1(j,2);
    w_wheel_dx(j+1,2) = output2(j,2);
end



% 1° grafico, metto gli input di velocità del rover nel primo caso

f(1) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time,speed(:,1),'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time,rot_speed(:,1),'-
b','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
ylabel('rover speed');
h_legend=legend('linear speed','rotational speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
```

```
saveas(f(1),'graf1.fig');

% 2°grafico, metto velocità angolari delle ruote, ossia gli output, nel
% primo caso

f(2) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time,w_wheel_sx(:,1),'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time,w_wheel_dx(:,1),'-
b','LineWidth',1.5,'MarkerSize',1.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
ylabel('wheels speed [rad/s]');
h_legend=legend('left wheels speed','right wheels
speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
saveas(f(2),'graf2.fig');

% 3°grafico, come il secondo ma in rpm invece che rad/s
f(3) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time,w_wheel_sx(:,1)*30/pi,'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time,w_wheel_dx(:,1)*30/pi,'-
b','LineWidth',1.5,'MarkerSize',1.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
```

```
hold off
xlabel('time [s]');
ylabel('wheels speed [rpm]');
h_legend=legend('left wheels speed','right wheels
speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
saveas(f(3),'graf3.fig');

% 4° grafico, metto gli input di velocità del rover nel secondo caso

f(4) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time,speed(:,2),'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time,rot_speed(:,2),'-
b','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
ylabel('rover speed');
h_legend=legend('linear speed','rotational speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
saveas(f(4),'graf4.fig');

% 5°grafico, metto velocità angolari delle ruote, ossia gli output, nel
% secondo caso

f(5) = figure();
```

```
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time,w_wheel_sx(:,2),'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time,w_wheel_dx(:,2),'-
b','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
ylabel('wheels speed [rad/s]');
h_legend=legend('left wheels speed','right wheels
speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
saveas(f(5),'graf5fig');

% 6°grafico, come il secondo ma in rpm invece che rad/s
f(6) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time,w_wheel_sx(:,2)*30/pi,'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time,w_wheel_dx(:,2)*30/pi,'-
b','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
ylabel('wheels speed [rpm]');
h_legend=legend('left wheels speed','right wheels
speed','Location','Best');
set(h_legend,'FontSize',22);
```

```
box on
axis square
saveas(f(6),'graf6.fig');


% come ultimo esempio riporto degli ingressi misto, sia in velocità
% rettilinea che di rotazione

time1 = [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22];

speed1 = zeros(23,1);
rot_speed1 = zeros(23,1);
w_wheel_sx1 = zeros(23,1);
w_wheel_dx1 = zeros(23,1);

for a= 1:1:numel(input3(:,1))
    speed1(a+1,1) = input3(a,1);
    rot_speed1(a+1,1) = input3(a,2);
    w_wheel_sx1(a+1,1) = output3(a,1);
    w_wheel_dx1(a+1,1) = output3(a,2);
end


f(7) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time1,speed1(:,1),'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time1,rot_speed1(:,1),'-
b','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
```

```
ylabel('wheels speed');
h_legend=legend('linear speed','rotational speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
saveas(f(7),'graf7.fig');

f(8) = figure();
set(gcf,'defaultaxesfontsize',28);
hold all
stairs(time1,w_wheel_sx1(:,1),'-
r','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','r','MarkerFaceCo
lor','w');
stairs(time1,w_wheel_dx1(:,1),'-
b','LineWidth',2.5,'MarkerSize',2.5,'MarkerEdgeColor','b','MarkerFaceCo
lor','w');
hold off
xlabel('time [s]');
ylabel('wheels speed [rad/s]');
h_legend=legend('left wheels speed','right wheels
speed','Location','Best');
set(h_legend,'FontSize',22);
box on
axis square
saveas(f(8),'graf8.fig');
```

Bibliografia

1. Wei Yu, Emmanuel Collins, Oscar Chuy, Dynamic modeling and power modeling of robotic skid-steered wheeled vehicles. Florida State University (USA).
2. S. Chiodini, M. Pertile, E. Bertolutti, R. Dalla Vecchia, D. Paganini, S. Debei, Morpheus: a field robotic testbed for soil sampling and autonomous navigation. Università degli studi di Padova, 2015.
3. Yutaka Kanayama, Yoshihiko Kimura, Fumio Miyazaki, Tetsuo Noguchi, A stabile tracking control method for an autonomous mobile robot, 1990.
4. Alberto Doria, Vittore Cossalter, Dinamica del veicolo, 2013.
5. Cristian secchi, Robotica mobile.
6. Krzysztof Kozlowski, Dariusz Pazderski, Modeling and control of a 4 whhel skid-steering mobile robot, Poznan University of Technology, 2004.
7. Tianmiao Wang, Yao Wu, Jianhong Liang, Chenhao Han, Jiao Chen and Qiteng Zhao, Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor, robotics institute, Beihang University, 2015.
8. Hongpeng Wang, Junjie Zhang, Jingang Yi, Dezhen Song, Suhada Jayasuriya, and Jingtai Liu, Modeling and motion stability analysis of skid-steered mobile robots.
9. Benjamin Shamah, Experimental comparison of skid-steering vs. explicit steering for a wheeled mobile robot, the robotics institute, Mellon University (USA), 1999.
10. Frantisek Solc, Jaroslav Sembera, kinetic modeling of a skid steered robot, international conference on signal processing, robotics and automation, University of Cambridge, 2008.
11. Anthony Mandow, Jorge L. Martinez, Jesus Morales, Jose L. Blanco, Alfonso Garcia-Cerezo and Javier Gonzalez, Experimental Kinematics for wheeled skid-steer mobile robots.
12. Riccardo Torlone, introduzione al C++, 2001
13. Riccardo Torlone, operazioni di input/output in C++, 2001
14. www.roverchallenge.eu

15. www.maxonmotor.it
16. www.udoo.it
17. www.atmel.com

Ringraziamenti

Desidero ringraziare il professor Debei per la possibilità che mi ha dato di svolgere una tesi sperimentale nell'ambito della robotica; inoltre ringrazio il professor Pertile per la disponibilità e per i consigli che mi ha dato durante questo periodo.
Un pensiero va alla mia famiglia, che mi ha sostenuto durante questi anni di studi, incitandomi a dare sempre il massimo, e a tutti gli amici.
Una dedica in particolare va ai miei nonni, che molto hanno contribuito durante tutto il mio percorso all'interno dell'Università.