Università degli Studi di Padova

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Elettronica

Progettazione e realizzazione di un Hexapod mediante cinematica inversa

Relatore:
Prof. Matteo Meneghini

Laureando:
Thomas Zamprogno
2008590

Anno Accademico: 2022/2023
Data di laurea: 27/09/2023

Abstract

Questa tesi analizza la progettazione e la realizzazione di un robot hexapod, un veicolo meccanico capace di camminare su sei zampe.Sarà fornita una descrizione dell'hardware utilizzato, con spiegazione dei vari componenti impiegati, insieme alle motivazioni che hanno guidato tali scelte.Successivamente, sarà affrontata la cinematica inversa utilizzata per il movimento del robot.
Verrà presentato e spiegato nei punti più importati il codice con il quale viene effettivamente implementata la cinematica.Infine, saranno presentati e discussi alcuni aspetti migliorativi che possono garantire un progetto più efficiente.

Indice

Capitolo 1

Introduzione

Al giorno d'oggi i sistemi robotici sono generalmente suddivisi in due grandi categorie: robot mobili e robot manipolatori. I primi vengono utilizzati prevalentemente in ambito industriale per manipolare e spostare materiali senza l'aiuto umano rimanendo fermi nella posizione in cui vengono installati. Al contrario i robot mobili sono dei dispositivi che hanno la capacità di spostarsi grazie al controllo remoto di un supervisore o in modo autonomo grazie a eventuali sensori installati. Concentrandosi nell'area dei robot mobili, questi possono avere generalmente due grandi suddivisioni a seconda del meccanismo che permette loro di muoversi nell'ambiente circostante, che sono :

-ruote;

-gambe.

La scelta tra le diverse configurazioni viene fatta in base all'obiettivo e all'applicazione per la quale vengono impiegati. I robot dotati di ruote sono ad oggi i più diffusi per la loro semplicità sia in fase di realizzazione sia in quella di manutenzione, tuttavia come svantaggio presentano la possibilità di muoversi solamente su superfici regolari e presentano pertanto l'impossibilità di superare ostacoli o grandi irregolarità della superficie su cui si muovono. Inoltre a causa delle ruote, con cui si muovono, lasciano solchi a terra, che in determinati casi possono rappresentare un problema. Invece, i robot dotati di gambe presentano il grande vantaggio di superare questi limiti permettendo loro di muoversi senza problemi in superfici che risultano molto irregolari. Inoltre è possibile in alcuni casi poter variare anche l'altezza del corpo a cui sono collegate le gambe, permettendo così di superare ostacoli sporgenti. Un altro vantaggio rispetto alla locomozione mediante ruote è la tolleranza a possibili guasti. Infatti nei robot dotati di ruote, il guasto a una di queste comporterebbe una perdita di mobilità

dell'intera struttura. Al contrario i veicoli dotati di gambe presentano in alcune configurazioni un numero elevato di gambe (fino a 8) e questo permette loro di mantenere l'equilibrio e di poter continuare la locomozione anche in presenza di una o più gambe danneggiate. Infine,come ultimo vantaggio, si ha la possibilità di poter utilizzare le gambe non solo con l'obiettivo di mobilitazione. Questo perché a causa della grande stabilità in assenza di una o più gambe, una può tranquillamente essere impiegata come manipolatore. Nonostante questi grandi vantaggi presentati, i robot dotati di gambe presentano alcune limitazioni rispetto a quelli implementati con ruote. Tra questi vi sono la velocità di movimento, ben inferiori a quella che si potrebbe ottenere con una medesima implementazione con ruote, e la difficoltà nel realizzare e progettare complessi algoritmi di controllo per la stabilizzazione e il movimento. In fine si nota la necessità di un numero di attrattori che aumenta notevolmente al crescere del numero di gambe che si vogliono implementare (ad esempio un robot con 6 gambe richiede 18 attuatori) con incremento dell'energia necessaria ad alimentarli. [1]
Nella seguente tesi verrà trattata la realizzazione di un robot che impiega una mobilitazione mediante gambe, più precisamente si tratterà la progettazione di un robot hexapod, che consiste in un robot dotato di 6 gambe. Nella realizzazione del robot verrà impiegata un struttura in alluminio preventivamente realizzata,nella quale verranno installati in totale 18 servo motori che ne permettono il movimento in base a determinati angoli calcolati in funzione dei movimenti voluti. Ogni singola gamba sarà dotata di tre servo che gli permettono di assumere diverse posizioni, si ottiene pertanto un robot dotato di 18 gradi di libertà (dof). Per il movimento del robot verranno impiegate due modalità: la prima permette al hexapod di muoversi in modo autonomo evitando eventuali ostacoli che possono intralciare il movimento, la seconda invece permette di controllarlo manualmente con ad esempio lo smartphone grazie all'impiego del modulo bluetooth. Inoltre viene inserito un led RGB che permette di segnale il diverso stato di carica della batteria Lipo impiegata. Per trattare in modo esaustivo quanto effettuato, nel secondo capitolo verranno presentati gli obiettivi che si sono posti nella realizzazione del robot con eventuali limiti annessi. Si proseguirà nel terzo capitolo con la presentazione del circuito realizzato con la spiegazione del hardware scelto per la progettazione. Nel quarto capitolo verrà presentata la teoria che sta alla base della gestione dei movimenti del robot nelle varie direzioni,con la relativa trattazione delle equazioni della cinematica inversa che sono state ricavate. In seguito verranno illustrate nel quinto capitolo le parti più rilevanti del codice che

realizzano quanto ricavato nel precedente capitolo, con anche la spiegazione delle parti riguardanti il controllo mediante bluetooth e il monitoraggio dello stato di carica della batteria Lipo impiegata. In conclusione verranno riportate nel sesto capitolo le considerazioni e il funzionamento ottenuto, mentre verranno affrontate possibili migliorie nel settimo capitolo. In appendice A verrà riportato l'intero codice realizzato mentre, per ultimo, in appendice B sarà riportato il layout PCB del circuito realizzato e le foto dell'esapode.

Capitolo 2

Specifiche del Progetto

2.1 Obiettivi realizzativi

Con questo progetto si è voluto realizzare un robot hexapod con l'impiego del microcontrollore Arduino. Questo è stato progettato in modo da rispettare le seguenti specifiche:

- Essere in grado di impiegare due diverse modalità di movimento a seconda della scelta effettuata dal utente. La prima mediante il controllo manuale grazie all'impiego di un modulo di comunicazione bluetooth, e la seconda mediante un funzionamento autonomo;
- Ricevere in input mediante modulo bluetooth un determinato comando di movimento ed essere in grado di interpretarlo e di posizionarsi come voluto;
- Essere in grado di muoversi con funzionamento autonomo senza sbattere contro eventuali ostacoli che ne possano impedire il movimento;
- Riuscire a calcolare le posizioni in gradi dei vari servomotori in base al movimento che si desidera attuare;
- Posizionare i vari servo in base agli angoli calcolati con la cinematica inversa;
- Monitorare lo stato di carica della batteria Lipo, segnalando il diverso stato di carica grazie all'impiego di un led Rgb;
- Regolare l'angolazione dei servo in un tempo non eccessivamente ridotto, con l'obiettivo di muovere il robot con una velocità tale che non ne comporti il danneggiamento;
- Rimanere in una posizione fissa in assenza di comandi di direzione.

2.2 Limiti di funzionamento

Nella realizzazione si sono assunti alcuni limiti in fase di progettazione a causa dei servomotori impiegati per permetterne il movimento.Questo perché per tale realizzazione sono stati scelti servo MG996R,che essendo servo di fascia medio-economica, non garantiscono una precisione accurata.Inoltre avendo una massima coppia di stallo pari a 11 Kg*cm,quando alimentati a 6V, non permettono di realizzare movimenti abbastanza fluidi che si potrebbero ottenere con servo di fascia più alta.Per tali motivi ci si aspetta dei movimenti non abbastanza "fluidi" con possibili imprecisioni negli angoli assunti a causa della bassa precisione dei sevo di fascia medio-economica.

Capitolo 3

Hardware impiegato

In questo capitolo verrà presentato l'hardware impiegato per la realizzazione.Come primo aspetto si analizza la struttura del robot,per passare poi al schema elettrico realizzato con conseguente analisi delle specifiche dei componenti impiegati.

3.1 Struttura del robot

La struttura impiegata per il robot è quella sotto riportata in figura 3.1. Questa è costituita da un corpo centrale al quale sono collegate le sei gambe che ne permettono il movimento.Ogni gamba è costituita da tre alloggi utilizzati per l'inserimento dei tre servo necessari a garantirne il movimento.Queste sono collegate alla struttura da un alloggio,che collegato al servo,permette di muovere la 4 gambe esterne con un'angolazione di circa 135° mentre le rimanenti 2 gambe interne con un'angolazione di 90°.

Ogni singola gamba è costituita come visibile in figura 3.2 da un pezzo che costituisce la Tibia della gamba, che posiziona il robot a contatto con il suolo.

Figura 3.1: Struttura hexapod

Questa è collegata all'intera struttura mediante il pezzo denominato come Femore, il quale permette di regolare l'altezza dal suolo assunta dalla base del robot variando l'angolazione del servo n.2.In fine l'ultimo elemento è la Coscia,che in tal caso è costituita solamente dall'alloggio del servo che collega la gamba alla piattaforma di base del robot,come precedentemente detto questo terminale permette di muovere l'intera gamba con un movimento semicircolare.Sempre in figura 3.2 sono riportati anche le possibili rotazioni che possono compiere i singoli servomotori.

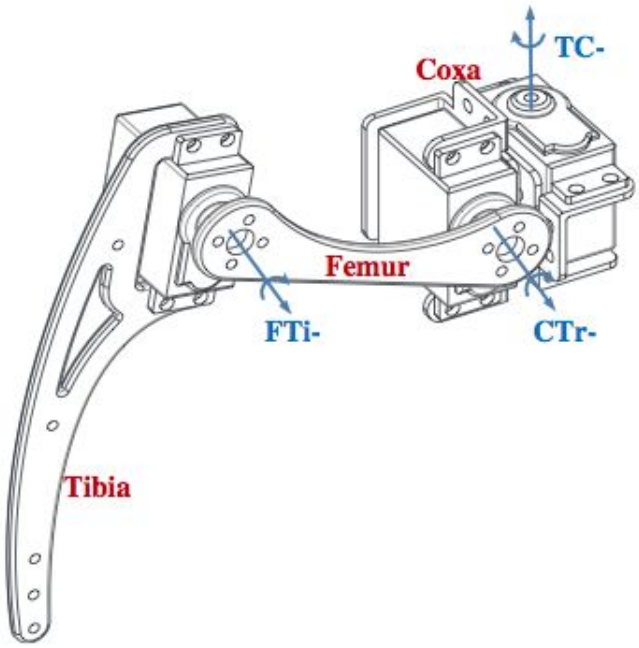

Figura 3.2: Struttura di una singola gamba

L'intera struttura impiegata è realizzata in alluminio di colore nero e le relative misure sono riportate in tabella 3.1

Pezzo	**Dimensioni(cm)**
Base	15.2 x 13.3
Femore	9
Tibia	12

Tabella 3.1: Dimensioni Hexapod

3.2 Schema elettrico

Nello schema riportato in figura 3.3, i pin di Arduino scelti per collegare i vari componenti stati identificati con la sigla `PIN_XX`.Per facilitare lo sbroglio della scheda PCB,si è scelto di separare i pin dedicati all'alimentazione dei servo da quelli di controllo.Per tale motivo gli header H1,H2 sono stati impiegati per collegare i pin di controllo del segnale di ciascun servomotore,mentre gli header identificati dalle sigle che vanno da H3 a H20 sono stati impiegati per l'alimentazione dei 18 servo.Per controllare l'accensione e lo spegnimento dell'alimentazione fornita dalla batteria Lipo,è stato inserito l'interruttore SW1, che interrompe sia la tensione in ingresso al buck sia quella per il monitoraggio dello stato di carica della batteria.Questo interruttore è stato in seguito inserito esternamente al PCB,fissandolo alla struttura.
Poiché la massima tensione che si può fornire in ingresso ai pin di Arduino è di 5V, si è deciso di ridurre la tensione fornita dalla batteria Lipo a un valore massimo di 4.2V,per mezzo del partitore realizzato mediante le resistenze R8 e R7.

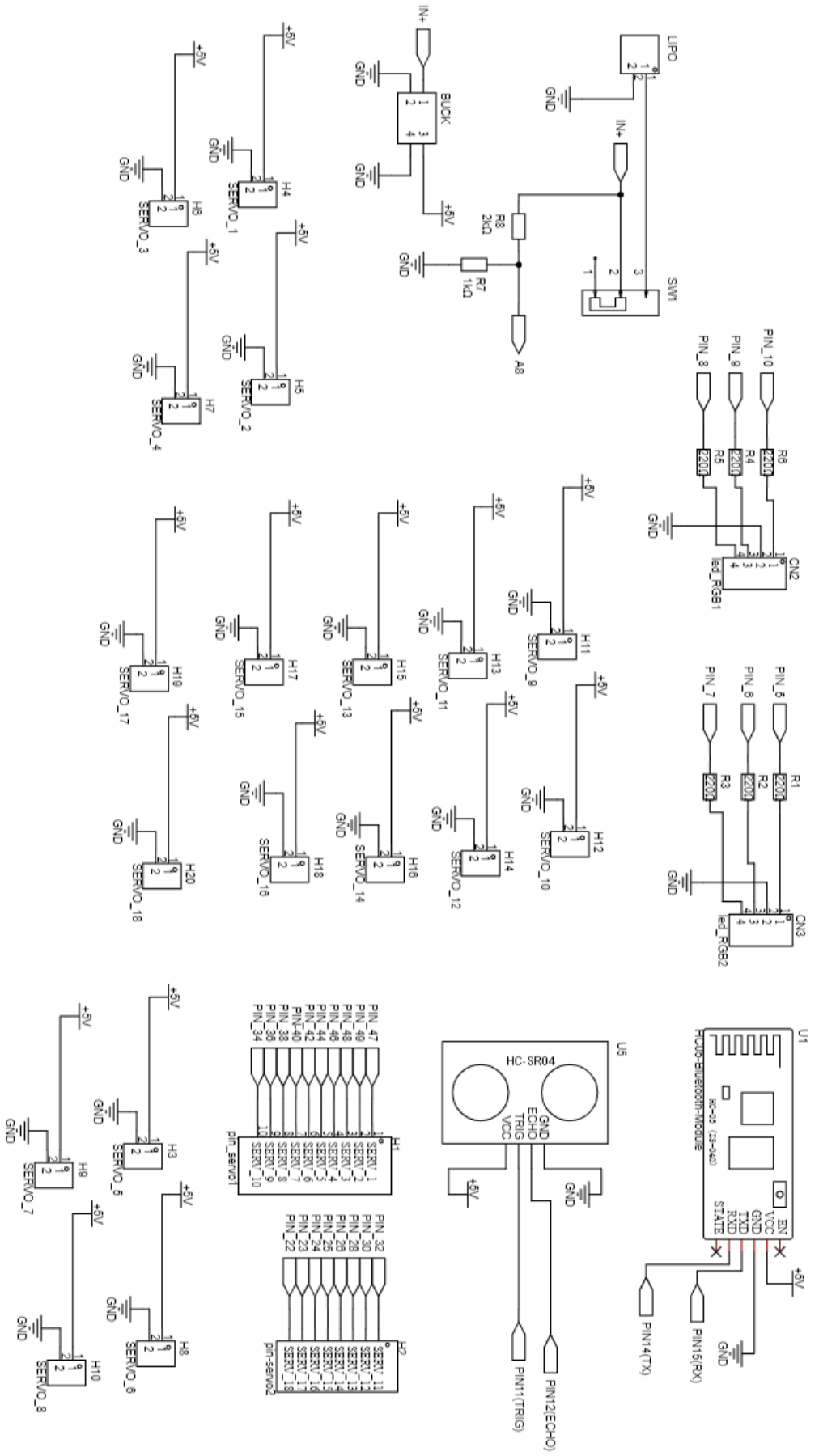

Figura 3.3: Schema elettrico

3.3 Servomotori

I servomotori sono motori elettrici rotanti che assumono una determinata posizione in base all'impulso di controllo che ricevono.Questi sono utilizzati soprattutto in progetti di robotica che richiedono azionamenti e sono costituiti da contenitori dai quali fuoriesce un piccolo albero solitamente metallico a cui collegare il braccio che riesce a ruotare mantenendo fissa la posizione raggiunta.All'interno sono essenzialmente costituiti da un motore CC,da una serie di ingranaggi che riducono il numero di giri, da un potenziometro e da una scheda di controllo.Sono solitamente dotati di tre fili (massa,alimentazione e uno per il controllo) che vengono distinti a seconda del colore.

3.3.1 Specifiche

I servomotori impiegati nel seguente progetto sono gli MG996R,che risultano compatibili con la struttura scelta in quanto rispettano le dimensioni imposte dagli alloggi.

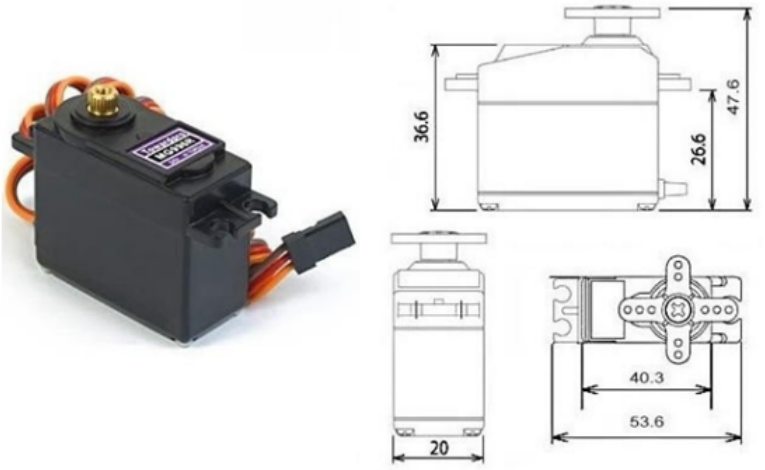

Figura 3.4: Dimensioni servo MG996R

Tali servo sono stati scelti anche per la massima coppia di stallo che possiedono, ovvero circa 9.4 Kg*cm con un'alimentazione di 4.8V. Le ulteriori specifiche più rilevanti considerando un'alimentazione di 5V sono riportate in tabella 3.2 mentre in figura 3.5 i collegamenti necessari. [2]

Dato	Valore
Coppia di stallo	$9.4\frac{Kg}{cm}$
Velocità operativa	$\frac{0.17}{60}\frac{secondi}{gradi}$
Corrente di funzionamento	500mA
Comando PWM	Da 1ms a 2ms
Tensione operativa	Da 4.8V a 7.2V

Tabella 3.2: Specifiche servo MG996R

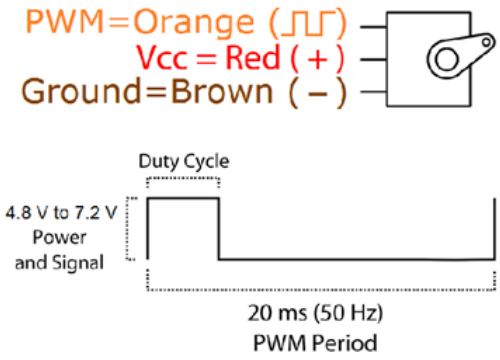

Figura 3.5: Collegamento servo e segnale PWM

3.3.2 Funzionamento

La tensione di alimentazione del servo impiegato può andare da 4.8V a 7.2V presentando,a seconda di questa,differenti valori di corrente di funzionamento nonché di massima coppia di stallo. Per settare l'angolo di questi motori si impiega un segnale di controllo PWM (Pulse width modulation),ovvero un segnale ad onda quadra nella quale viene fatta variare la durata del segnale a livello logico alto rispetto al periodo totale.Questo viene stabilito a seconda dell'angolo che si vuole far assumere al servomotore.La frequenza con cui viene inviato ogni segnale è di 50Hz, il che vuol dire che ogni periodo presenta una durata di 20ms. Durante questo periodo la massima durata del segnale a livello logico alto che si può applicare è di 2ms con i servo impiegati (MG996R). Come visibile in figura 3.6, in corrispondenza a un segnale a livello logico alto di durata 1ms il motore si posiziona ad un angolo di 0°, mentre ad una massima durata di 2ms si porta al angolo massimo che può raggiungere ovvero 180°.Di conseguenza variando in modo proporzionale la durata dell'impulso tra 1ms e 2ms è possibile variare l'angolo da 0° a 180°.Per ricavare la durata del impulso in funzione del angolo voluto si può impiegare la seguente equazione matematica: *Angolo in*

gradi=180(durata dell'impulso-1ms)* [3]

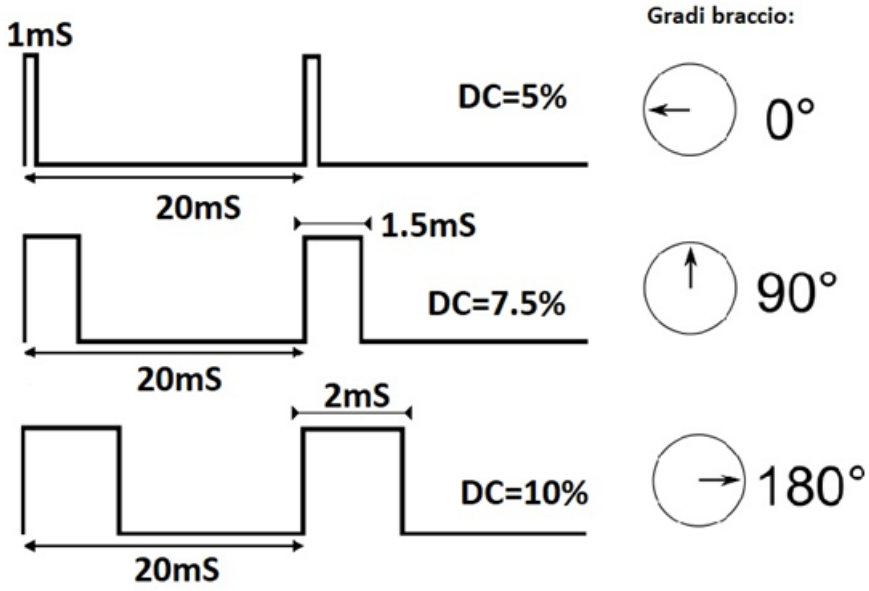

Figura 3.6: Relazione tra angolo e durata impulso

3.4 Convertitore Buck

Il convertitore Buck è un dispositivo regolabile in grado di convertire il valore di tensione fornita in ingresso in un valore distinto in uscita.A seconda che quest'ultima sia più elevata o più bassa di quella fornita in ingresso si distinguono due diversi convertitori, quello step-down in cui la tensione in uscita è più bassa e quello step-up che invece la alza.Verrà analizzato in questo caso la prima tipologia,in quanto impiegata per ridurre la tensione fornita dalla batteria Lipo da 12.6V a 5V.

3.4.1 Principio di funzionamento

Ogni alimentatore Buck è in grado di funzionare grazie a un transistor (BJT o MOSFET),un induttore e all'impiego della tecnica PWM. Il transistor viene impiegato come interruttore ON/OFF lavorando pertanto in saturazione o interdizione.In particolare in tale configurazione si ha il vantaggio di avere una dissipazione teorica nulla sul componente in quanto, essendo la potenza il prodotto tra tensione e corrente, in entrambi i stati di funzionamento uno dei due contributi risulta sempre nullo.

La modulazione PWM viene impiegata per far commutare tra ON e OFF il transistor utilizzato come interruttore.Variando tra ON e OFF si applica alternativamente tensione in uscita ottenendo così un valore medio che corrisponde al valore di tensione voluto.Con questa tecnica viene di solito impiegato un segnale ad onda quadra dotato di frequenza fissa nel quale viene fatto variare il segnale a livello logico alto, che corrisponde al tempo in cui viene fornita tensione al carico.

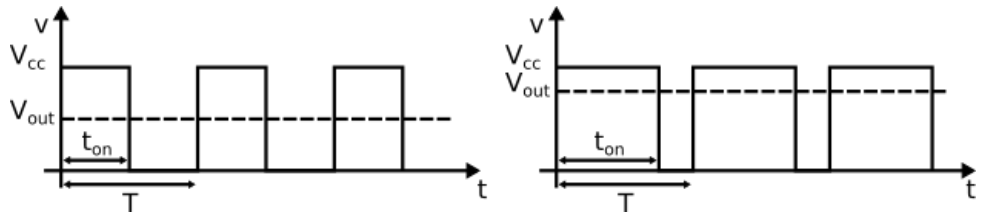

Figura 3.7: Regolazione Ton su periodo T

La relazione tra il valore il tensione in uscita e il valore di t_{ON} è dato da:

$$Vout = \frac{ton}{T} * VCC.$$

Lo schema base del convertitore step-down impiegato è riportato in figura 3.8.La regolazione della tensione in uscita è affidata a un sistema che regola il t_{ON} per aumentarla o diminuirla attraverso l'uso dell'interruttore S1 che in realtà è costituito da un transistor.

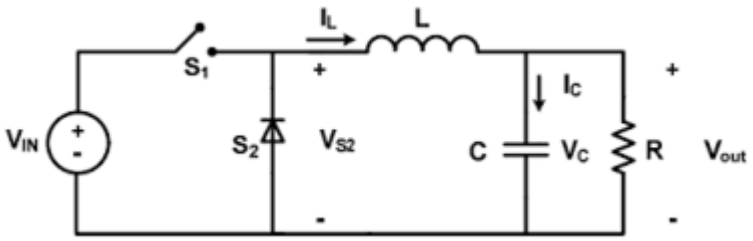

Figura 3.8: Schema base convertitore Buck step-down

Il funzionamento viene di seguito analizzato nel caso in cui l'interruttore sia chiuso o aperto,riportando in figura 3.9 il percorso compiuto dalla corrente nei due distinti casi. [4] Nella seguente trattazione viene analizzato il funzionamento in CCM, in cui viene imposta una corrente nell'induttore sempre maggiore di zero.Durate il t_{ON},l'interruttore è chiuso e in uscita è presente una tensione Vo.La corrente scorre dall'ingresso al carico attraversando l'induttore e il condensatore che accumulano energia mentre il diodo risulta off.Al termine del t_{ON}, l'interruttore viene aperto e l'energia immagazzinata dal condensatore e dall'induttore viene scaricata attraverso il carico.La corrente dopo aver attraversato il carico si richiude grazie al diodo di ricircolo che risulta attivo.

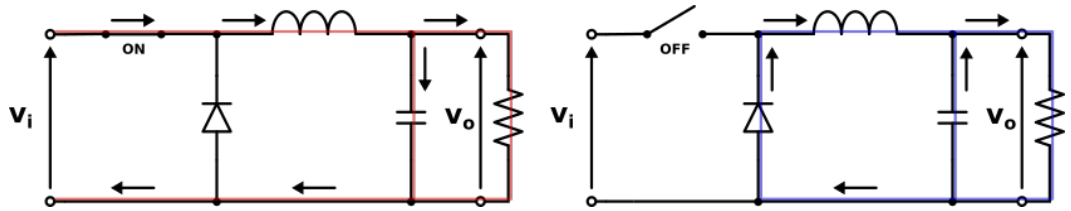

Figura 3.9: Andamento della corrente nello stato ON e OFF dell'interruttore

La tensione presente ai capi dell'induttore varia a seconda dei due distinti casi di funzionamento.Durante la fase ton, la tensione ai capi dell'interruttore è teoricamente Vin-Vo mentre durante il t_{OFF} la tensione è –Vo.Poiché la corrente che scorre nell'induttore dipende dall'integrale della tensione ai suoi capi,integrando l'andamento dell'onda rettangolare presente ai capi dell'induttore si ottiene l'andamento della corrente che vi scorre,che presenta una forma triangolare come visibile in figura 3.10

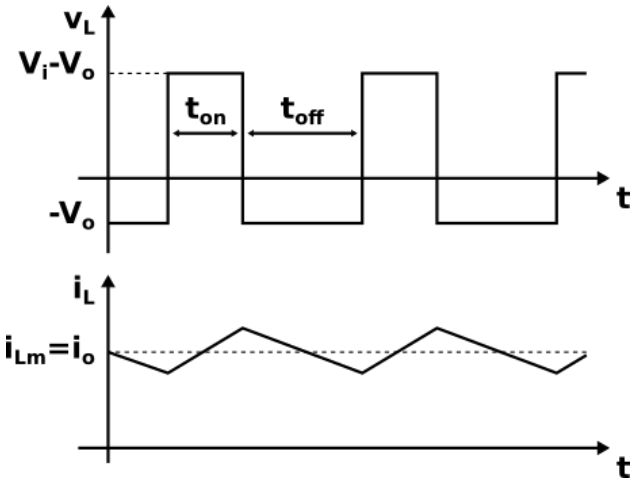

Figura 3.10: Andamento tensione e corrente dell'induttore

La variazione di corrente presente durante la fase t_{ON} è $\triangle$I_ON $= \frac{Vin-Vo}{L} * Ton$, mentre quella durante la fase t_{OFF} è $\triangle$I_OFF $= \frac{-Vo}{L} * Toff$,essendo nulla il termine Vin. La variazione di energia dell'induttore porta all'instaurarsi di un ripple nella tensione ai capi del condensatore, che presenta un andamento ondulatorio. Tale variazione di tensione risulta essere:

$$\triangle Vc = \frac{1}{C}\int i_C dt = \frac{1}{C}\int (i_L - i_O)dt \tag{3.1}$$

Da cui è possibile ottenere:

$$\triangle Vc = \frac{1}{2C}\left(I_C max\frac{T_o n + T_o ff}{2}\right) = \frac{1}{2C}\left(\frac{\triangle I_L}{2}\right)\left(\frac{T}{2}\right) = \left(\frac{\triangle I_L}{8fC}\right) \tag{3.2}$$

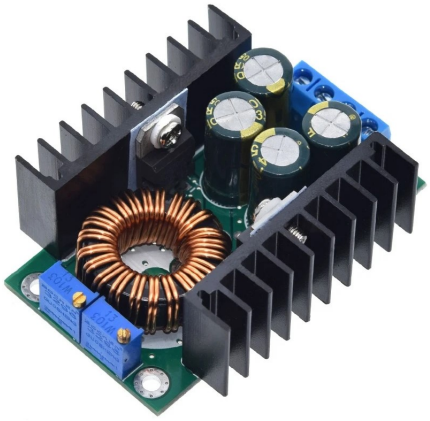

Figura 3.11: Convertitore Buck step-down impiegato

Il convertitore Buck step-down impiegato è riportato in figura 3.11, questo permette di ottenere una tensione in uscita regolabile per mezzo di un trimmer tra 0.8-28V con una massima corrente regolabile tra 0.2 e 9A in funzione di una tensione in ingresso compresa tra 7V e 32V.

3.5 Modulo Bluetooth

Lo standard Bluetooth è uno standard di trasmissione dati per reti personali senza fili WPAN (Wireless Personal Area Network).
Esso sfrutta onde radio ad una frequenza compresa tra i 2.4GHz e i 2.485GHz grazie l'impiego di un chip.La rete Bluetooth sfrutta un'architettura del tipo master/slave e basa il suo funzionamento su etichette a commutazione di pacchetto con uno scambio di dati a intervalli regolari.
Nel caso in esame viene impiegato per far comunicare il processore della scheda Arduino con lo smartphone che controlla il robot.

3.5.1 Specifiche

Il modulo bluetooth impiegato, riportato in figura 3.12,è identificato dalla sigla HC-05 e presenta una portata di 10 metri.Il vantaggio di tale modulo al rispetto del modello HC-06 è quello di poter essere programmabile sia come master che come slave.
Il modulo può essere disponibile sia come componente singolo oppure si può optare,come in questo caso,per schede in cui oltre ad essere già installo il modulo viene

fornito anche un regolatore di tensione,in quanto la tensione di alimentazione e di funzionamento del modulo è di 3.3V.La scheda impiegata è dotata di sei pin:

- **EN:** se collegato allo stato LOW disattiva il modulo, mentre qualora sia messo HIGH o non connesso permette il normale funzionamento del modulo;
- **VCC:**pin alimentazione del modulo che può essere compresa tra 3.6V e 6V;
- **GND;**
- **TXD:**uscita seriale del modulo, che necessità di essere collegato al pin RX di Arduino;
- **RXD:**ingresso seriale del modulo, che necessità di essere collegato al pin TX di Arduino;
- **STATE:**connesso al led2 del modulo che passo dallo stato LOW ad HIGH quando il modulo è collegato via bluetooth. [5]

Nel seguente progetto il pin di STATE e di EN non sono stati collegati,mentre il pin di alimentazione,ovvero VCC, è stato collegato a una tensione di 5V.

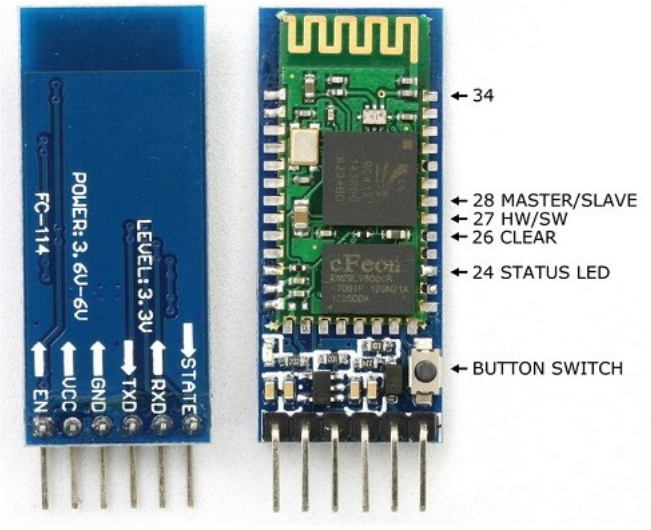

Figura 3.12: Modulo bluetooth HC-05 impiegato

3.6 Sensore ad ultrasuoni

I sensore ad ultrasuoni permettono,attraverso l'emissione di una sequenza di impulsi ad ultrasuoni,di determinare la distanza di eventuali oggetti posti in posizio-

ne frontale.Tali sensori possono essere impiegati in molti campi quali ad esempio l'impiego nei sensori di assistenza al parcheggio nelle automobili.

3.6.1 Specifiche

Il sensore ad ultrasuoni impiegato,e riportato in figura 3.13,è il modello HC-SR04,dotato di 4 pin: VCC,Trigger,Echo e GND.
Il pin di Trigger ha il compito di generare l'impulso ultrasonico mentre il pin di Echo permette di rilevare il segnale ultrasonico di ritorno.L'alimentazione impiegata in tale caso è di 5V.Nel dettaglio le principali caratteristiche tecniche del sensore sono riportate in tabella 3.3. [6]

Figura 3.13: Sensore HC-SR04 impiegato

Dato	**Valore**
Intervallo di misura	da 2cm a 400cm
Risoluzione	0.3cm
Corrente di funzionamento	15mA
Larghezza impulso di Trigger	10μs
Tensione operativa	5V

Tabella 3.3: Principali specifiche tecniche del modulo HC-SR04

3.6.2 Principio di funzionamento

I sensori ad ultrasuoni emettono impulsi ad ultrasuoni che si propagano in un fascio a forma di cono impiegando un dispositivo vibrante, il trasduttore, che genera l'onda ultrasonica. La frequenza con cui tale componente vibra, identifica la portata della distanza massima raggiungibile dal dispositivo. Più precisamente le onde sonore riescono a raggiungere distanze sempre più grandi al diminuire della frequenza, al contrario aumentando la frequenza si diminuisce la massima portata che queste onde possono avere.Il sensore permette di generare un treno

di impulsi ad ultrasuoni, che saranno propagati nell'ambiente circostante e incontrando un ostacolo torneranno indietro verso il sensore che li ha emessi.Quando il sensore rileva l'eventuale onda di ritorno porterà allo stato logico basso il pin di Echo precedentemente portato allo stato logico alto con l'invio dell'impulso. Sfruttando l'intervallo di tempo che intercorre tra l'emissione del treno di impulsi e il suo ritorno è possibile calcolare la distanza a cui è posto l'oggetto.
Per determinare la distanza a cui è posto l'eventuale oggetto si impiega la velocità del suono che in aria, è di 343 m/s. Per l'impiego effettuato,in cui è necessario riconoscere oggetti posti a centimetri di distanza,tale valore viene portato in cm/microsecondi ottenendo 0.0343 cm/μs. Sfruttando la formula della velocità è possibile ricavare lo spazio percorso durante l'intervallo temporale misurato. Risulta perciò $Spazio(incm) = v * t = 0.0343 * t(\mu s)$. Tuttavia tale valore è comprensivo sia del tempo necessario a raggiungere l'ostacolo sia di quello necessario per ritornare al sensore che l'ha generato,perciò l'intervallo di distanza necessario è pari alla metà,ottenendo $Spazio(incm) = 0.01755 * t(\mu s)$ o come riportato nel datasheet, $Spazio(incm) = t(\mu s)/58$. [6]
Per rilevare la distanza è necessario pertanto inviare un impulso di 10ms sul pin di Trigger del sensore e calcolare il tempo che intercorre tra la generazione del segnale e l'effettivo segnale di ritorno,ovvero quando il segnale di Echo si porta a livello basso.In figura 3.14 è riportato il diagramma temporale dei segnali.

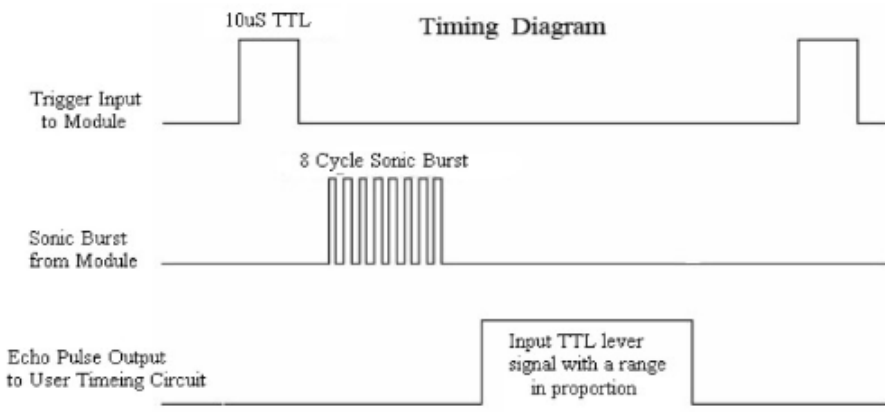

Figura 3.14: Diagramma temporale dei segnali

3.7 Arduino

La scheda Arduino impiegata per la seguente realizzazione è una scheda open-source cioè con licenza libera, impiegata per realizzare progetti di elettronica e

robotica. In commercio si trovano numerose versioni a seconda del numero di pin disponibili e del microcontrollore che vi è installato. Nella seguente applicazione viene impiegata la scheda Arduino Mega 2560,il cui pinout viene riportato in figura 3.15.
Questa è stata scelta in quanto dispone di un numero di pin sufficienti a controllare i 18 servomotori necessari a garantire il movimento del robot.La scheda è costituita da un processore AT-mega2560 che,con l'oscillatore in cristallo installato, permette di avere una frequenza di clock di 16MHz,avendo inoltre un memoria Flash di 256KB.Grazie al processore più potente rispetto alla scheda Arduino Uno,ha la possibilità di controllare 54 pin I/O digitali,di cui 15 permettono di offrire un'uscita controllata in PWM,e 16 pin analogici. [7] La scheda può essere alimentata grazie al connettore jack oppure, come in questo progetto,l'alimentazione può essere fornita direttamente dal pin Vin,con l'attenzione che il valore di questa risulti di 5V,non essendo presente un regolatore che possa ridurne il valore.Per poter programmare la scheda si fa uso del software Arduino IDE,un ambiente di sviluppo che permette di scrivere il codice necessario e di poterlo inseguito caricare sulla scheda.Per la programmazione si fa uso di un linguaggio di programmazione di Arduino,che è basato su C++,ma notevolmente semplificato.
Per controllare i 18 servo con Arduino si è deciso di impiegare la libreria Servo.h,che permette di utilizzare tutti i pin digitali e analogici di Arduino e non solo i pin PWM,questo permette ad Arduino Mega di essere in grado di controllare fino a 48 servomotori.

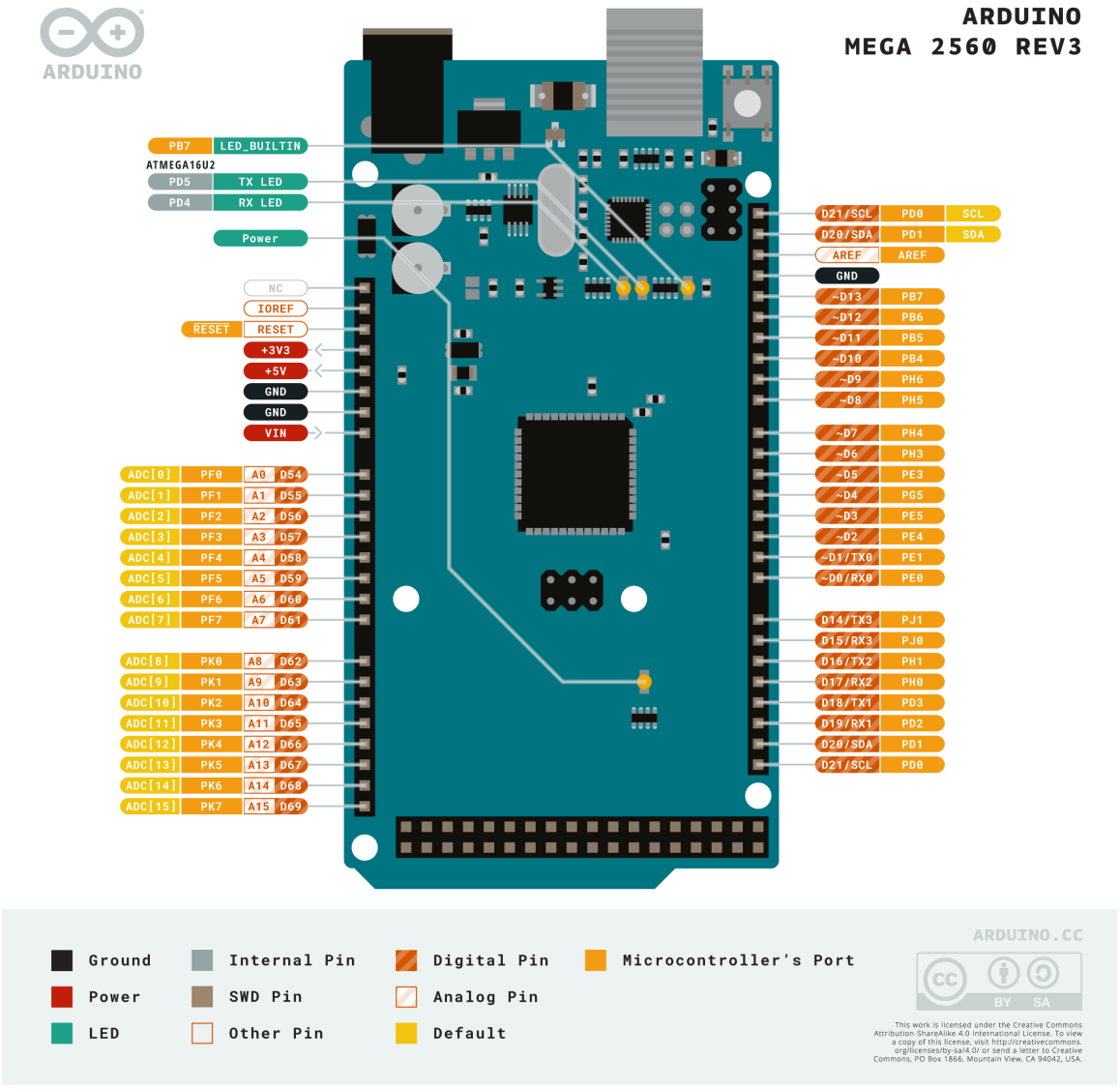

Figura 3.15: Pinout Arduino Mega 2560

Capitolo 4

Teoria del movimento

In questo capitolo viene illustrata la teoria che sta alla base del movimento del robot.

4.1 Schema del movimento

In questo capitolo viene descritta la locomozione del hexapod.Per permettere al robot di muoversi ci si è ispirati al movimento che compiono i ragni nella loro fase di camminata, i quali impiegano un gait a "tripod".

4.1.1 Tripod Gait

Il "tripod gait" (andatura a tre punti di appoggio) è un tipo di movimento utilizzato da molti robot esapodi per camminare e simula il modo in cui gli insetti a sei zampe si muovono.Per capire come funziona il tripod gait,così come altre tipologie di andature,si può inizialmente considerare la locomozione compiuta da una singola zampa.Il movimento che compie è visibile in figura 4.1 e può essere suddiviso in due fasi che avvengono in un singolo ciclo di passo:

- Fase di appoggio (stance phase):durante questa fase la gamba è a contatto con il suolo,dove supporta e spinge il corpo.Durante questa fase la zampa si muove verso la parte posteriore/anteriore del corpo determinando il movimento in avanti o all'indietro del hexapod;

- Fase di movimento (swing phase):durante questa fase la gamba viene sollevata dal terreno e si sposta verso la posizione iniziale della successiva fase di appoggio.Durante questa fase la zampa si muove verso la parte anterio-

re/posteriore del corpo determinando il movimento in avanti o all'indietro del hexapod.

Nel caso di un tripod gait,ogni fase avviene per la metà di un periodo di ciclo di passo. [8]

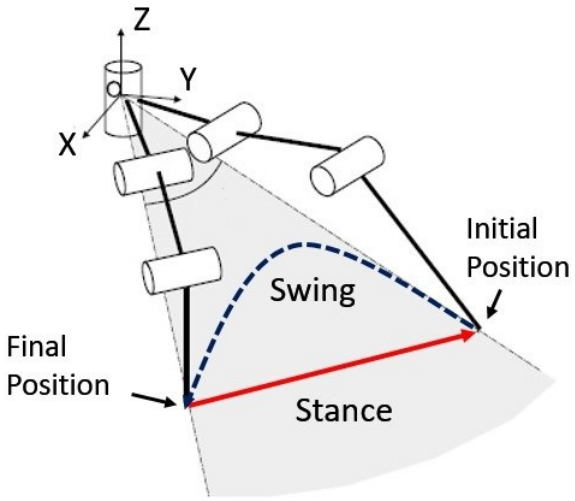

Figura 4.1: Fasi di locomozione

Durante il tripod gait,il robot utilizza pertanto un gruppo di tre zampe come punti di appoggio (stance phase) mentre le altre tre zampe si sollevano e avanzano (swing phase),così facendo si hanno sempre tre zampe a contatto con il suolo che permettono di formare un triangolo garantendo la stabilità dell'intera struttura.In dettaglio è possibile pertanto distinguere due gruppi di zampe che si muovo alternando la fase di avanzamento e di supporto.Nel hexapod presente in figura 4.2 si ha pertanto un gruppo A costituito dalle zampe 1,3,5 e un gruppo B identificato dalle zampe 4,6,2.

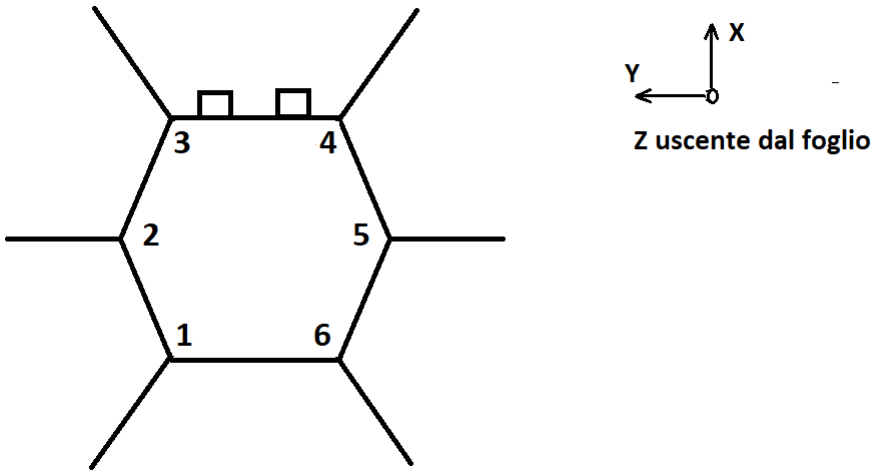

Figura 4.2: Modello Gait del hexapod

La sequenza con cui si alternano le zampe è riportato nello schema in figura 4.3.Nel dettaglio, il robot inizia con tre zampe di un gruppo che sono a contatto con il terreno, chiamate zampe di supporto.Queste zampe sostengono il peso del robot e forniscono la stabilità necessaria.Nel frattempo, le altre tre del gruppo opposto,chiamate zampe di avanzamento,si sollevano e si spostano in avanti per raggiungere una nuova posizione.
Una volta che le zampe di avanzamento hanno raggiunto una nuova posizione,le zampe di supporto si sollevano e diventano zampe di avanzamento,mentre le precedenti zampe di avanzamento diventano le nuove zampe di supporto.Questo processo viene ripetuto in modo ciclico per consentire al robot di avanzare.

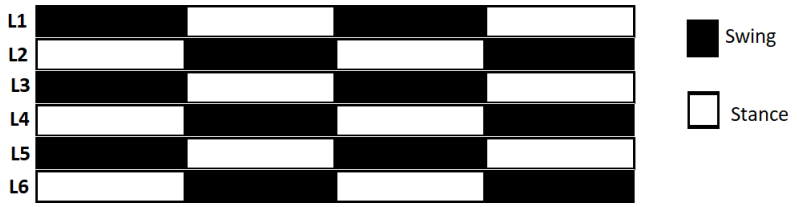

Figura 4.3: Sequenza movimento delle gambe

Nel seguito verranno trattati distintamente i 4 movimenti base necessari a garantire il movimento dell'esapode nelle quattro direzioni. [12]
Per muovere l'esapode in avanti,come visibile in figura 4.4,è possibile distinguere due fasi che si ripetono:
Fase 1:

- Le zampe che costituiscono il gruppo A, ovvero la quella anteriore sinistra(3),centrale destra(5) e posteriore sinistra(1) si sollevano e si spostano

in avanti per raggiungere la nuova posizione (swing phase);

- Le zampe del gruppo B, ovvero le zampe anteriore destra(4),centrale sinistra(2) e posteriore destra(6) sono a contatto con il terreno e svolgono il ruolo di zampe di supporto spostandosi all'indietro (stance phase).

Fase 2:

- Le zampe che costituiscono il gruppo A,diventano ora le nuove zampe di supporto spostandosi indietro a contatto con il suolo (stance phase);

- Le zampe del gruppo B,invece si sollevano e si spostano in avanti per raggiungere la nuova posizione (swing phase).

Ogni fase di questo specifico movimento,così come quelle successive per i restanti tre movimenti,sono state eseguite con la specifica che al termine di un ciclo di movimento (ovvero la somma del periodo di stance e swing),l'intero hexapod si sia spostato dalla sua posizione precedente di 80mm.

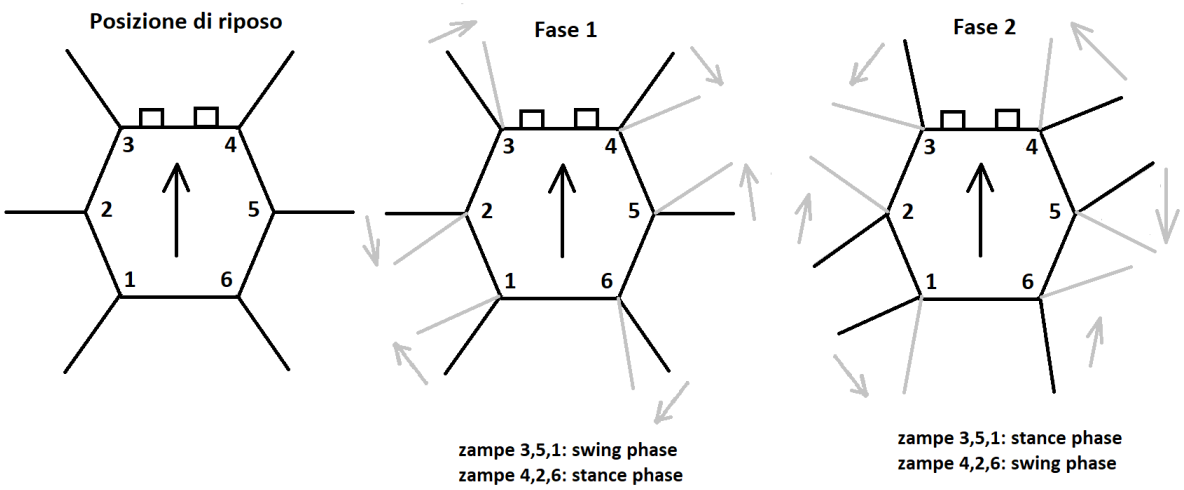

Figura 4.4: Sequenza per permettere il movimento in avanti. In nero la posizione delle zampe nella fase precedente,mentre in grigio quella assunta nella nuova. Ripetendo la fase 1 e 2 si ottiene il movimento in avanti.

Per garantire il movimento dell'esapode all'indietro,è necessario compiere la swing phase e la stance phase nella direzione opposta a quella effettuata nel movimento in avanti.Si distinguono pertanto,come nel precedente movimento,le due fasi:
Fase 1:

- Le zampe che costituiscono il gruppo A, ovvero la quella anteriore sinistra(3),centrale destra(5) e posteriore sinistra(1) si sollevano e si spostano all'indietro per raggiungere la nuova posizione (swing phase);
- Le zampe del gruppo B, ovvero le zampe anteriore destra(4),centrale sinistra(2) e posteriore destra(6) sono a contatto con il terreno e svolgono il ruolo di zampe di supporto spostandosi in avanti (stance phase).

Fase 2:

- Le zampe che costituiscono il gruppo A,diventano ora le nuove zampe di supporto spostandosi in avanti a contatto con il suolo (stance phase);
- Le zampe del gruppo B,invece si sollevano e si spostano all'indietro per raggiungere la nuova posizione (swing phase).

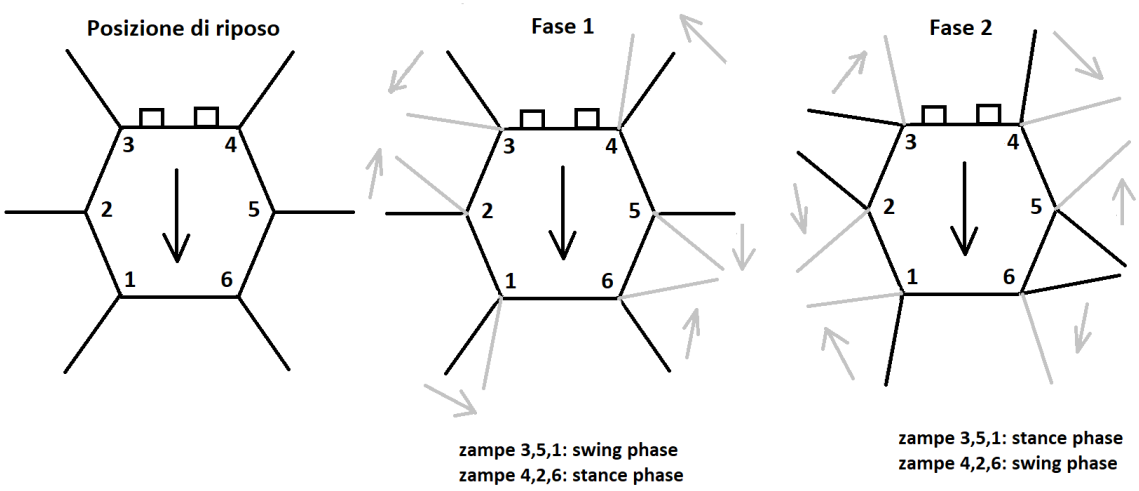

Figura 4.5: Sequenza per permettere il movimento all'indietro.In nero la posizione delle zampe nella fase precedente,mentre in grigio quella assunta nella nuova.Ripetendo la fase 1 e 2 si ottiene il movimento all'indietro.

Per permettere all'esapode di girare a destra e a sinistra è necessario che le zampe di ogni lato si muovano nella stessa direzione.Questo porta perciò ad avere le zampe centrali di ogni gruppo a muoversi in direzione opposta a quelle anteriori e posteriori del medesimo gruppo.Per quanto concerne alla rotazione del hexapod a sinistra,si individuano,come fatto in precedenza,due fasi:
Fase 1:

- Le zampe che costituiscono il gruppo A, ovvero la quella anteriore sinistra(3) e posteriore sinistra(1) si sollevano e si spostano all'indietro per raggiungere

la nuova posizione, mentre la zampa centrale destra(5) esegue lo stesso movimento però in avanti(swing phase);

- Le zampe del gruppo B, ovvero le zampe anteriore destra(4)e posteriore destra(6) sono a contatto con il terreno e svolgono il ruolo di zampe di supporto spostandosi all'indietro,mentre la zampa centrale sinistra(2) esegue lo stesso movimento spostandosi però in avanti(stance phase).

Fase 2:

- Le zampe che costituiscono il gruppo A,diventano ora le nuove zampe di supporto.Le zampe 3 e 1 si spostano in avanti a contatto con il suolo, mentre la zampa numero 5 esegue il medesimo movimento però all'indietro (stance phase);

- Le zampe 4 e 6 del gruppo B,invece si sollevano e si spostano in avanti per raggiungere la nuova posizione,mentre la zampa numero 2 esegue lo stesso movimento ma all'indietro (swing phase).

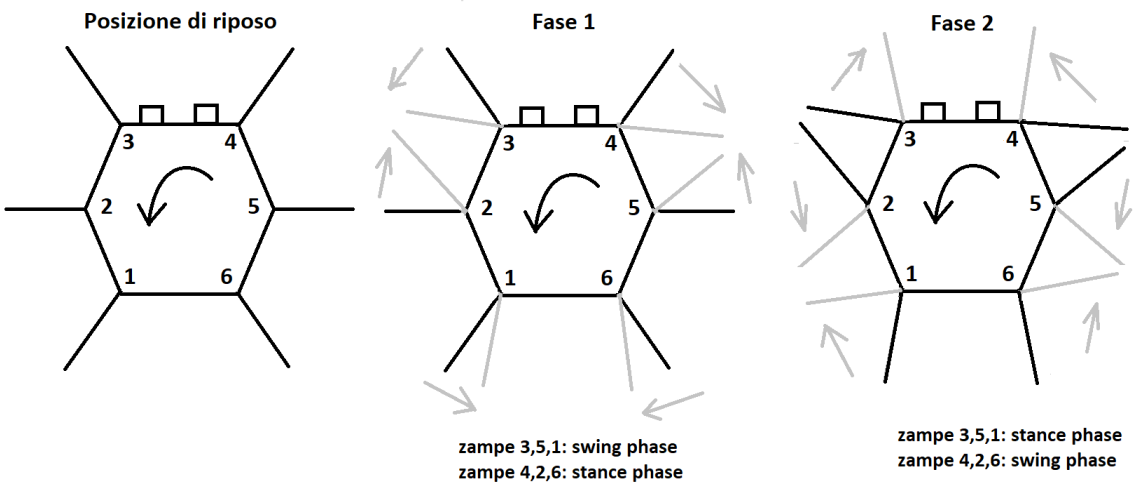

Figura 4.6: Sequenza per permettere il movimento a sinistra.In nero la posizione delle zampe nella fase precedente,mentre in grigio quella assunta nella nuova. Ripetendo la fase 1 e 2 si ottiene la rotazione desiderata.

In ultima vengono presentate le due fasi che permettono la rotazione a destra.
Fase 1:

- Le zampe che costituiscono il gruppo A, ovvero la quella anteriore sinistra(3) e posteriore sinistra(1) si sollevano e si spostano in avanti per raggiungere la nuova posizione, mentre la zampa centrale destra(5) esegue lo stesso movimento però all'indietro(swing phase);

- Le zampe del gruppo B, ovvero le zampe anteriore destra(4)e posteriore destra(6) sono a contatto con il terreno e svolgono il ruolo di zampe di supporto spostandosi in avanti,mentre la zampa centrale sinistra(2) esegue lo stesso movimento spostandosi però all'indietro(stance phase).

Fase 2:

- Le zampe che costituiscono il gruppo A,diventano ora le nuove zampe di supporto.Le zampe 3 e 1 si spostano all'indietro a contatto con il suolo, mentre la zampa numero 5 esegue il medesimo movimento però in avanti(stance phase);

- Le zampe 4 e 6 del gruppo B,invece si sollevano e si spostano all'indietro per raggiungere la nuova posizione,mentre la zampa numero 2 esegue lo stesso movimento ma in avanti(swing phase).

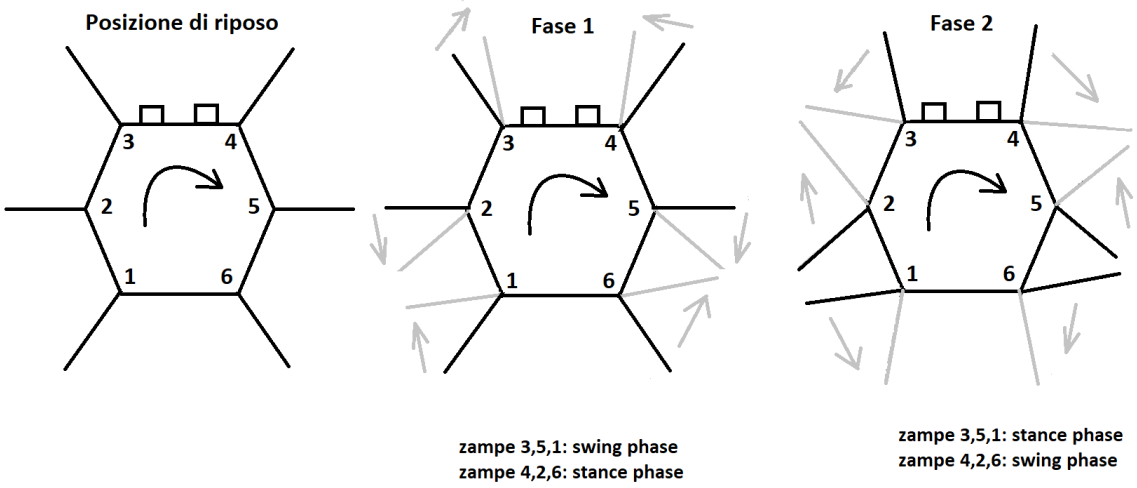

Figura 4.7: Sequenza per permettere il movimento a destra. In nero la posizione delle zampe nella fase precedente,mentre in grigio quella assunta nella nuova. Ripetendo la fase 1 e 2 si ottiene la rotazione desiderata.

4.1.2 Traiettoria delle zampe

La traiettoria nel movimento delle zampe di un robot si riferisce al percorso seguito dalle zampe durante il loro movimento.Nel contesto dei robot esapodi,la traiettoria delle zampe è importante per determinare come il robot si sposta.
Nella progettazione della traiettoria che le zampe del robot devono compiere si distinguono due diverse forme che vengono compiute nella fase di swing e di stance.Esistono diversi modelli di traiettoria utilizzati per descrivere il movimento

delle zampe degli hexapod,ma in questo caso si è deciso di impiegare una traiettoria ellittica per la fase di swing e una traiettoria rettilinea per la fase di stance. La traiettoria ellittica,impiegata nella fase di movimento, è una forma di traiettoria che segue una curva ellittica.Nell'ambito del movimento delle zampe degli hexapod,una traiettoria ellittica implica che la zampa segua un percorso a forma di ellisse durante il suo movimento,anziché un percorso lineare o curvilineo.Questo tipo offre diversi vantaggi tra cui i più importanti sono:

- *Efficienza:* la traiettoria ellittica consente al hexapod di coprire distanze maggiori con un minor sforzo.L'ellisse offre una combinazione ottimale di movimento orizzontale e verticale, riducendo al minimo l'energia richiesta per il movimento delle zampe;

- *Stabilità:* la traiettoria ellittica permette un movimento più stabile e bilanciato.L'ellisse offre un percorso controllato per la zampa,riducendo al minimo il rischio di oscillazioni indesiderate o di instabilità durante la camminata.

Invece la traiettoria rettilinea,impiegata nella fase di stance, è una forma di traiettoria che segue una linea retta ed è utilizzata in questo caso per unire il punto finale della fase swing con quello iniziale.
L'immagine 4.8 illustra dall'alto la traiettoria compiuta da ogni zampa durante la fase di stance e di swing.Si è deciso di fissare per comodità un piano XYZ per ogni zampa,avente origine nel punto di collegamento tra la zampa e il corpo dell'esapode.

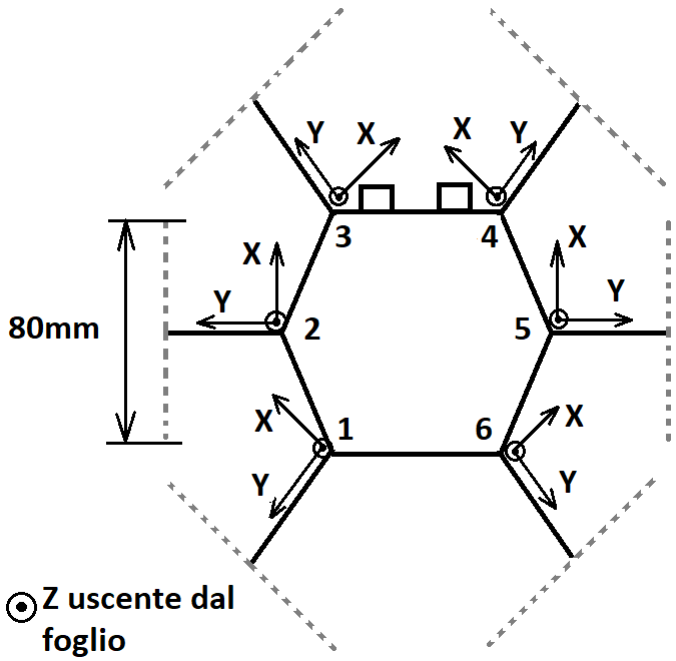

Figura 4.8: Traiettoria compiuta da ogni zampa.

Le espressioni per la semi-curva ellittica della fase di swing,nel piano XYZ fissato,sono riportate di seguito. [9] [10]
Considerando il sistema cartesiano scelto,una volta fissato un offset fisso per l'asse Y,corrispondente circa alla lunghezza U della zampa nella posizione di riposo(vedere figure 4.10 e 4.11),la traiettoria del punto finale di ogni zampa giace su un piano XZ.Così facendo le curve caratteristiche della traiettoria nel piano P=(X,Y_{offset},Z) sono:

$$X = \frac{L}{2} cos\varphi \tag{4.1}$$

$$Z = \frac{H}{2} sin\varphi \tag{4.2}$$

dove L,H,φ sono rispettivamente la lunghezza dello step,l'altezza dello step e l'angolo polare compreso fra 0 e π.
Invece la traiettoria nella fase di stance,nel piano XYZ fissato,è costituita da un'unica equazione che identifica il movimento della parte finale della zampa lungo una retta a contatto con il suolo.Ciò corrisponde in questo caso,oltre a un offset di Y,anche a un valore fisso di Z(Z=Z_{offset},scelto in base all'altezza dell'esapode nella posizione di riposo)tale da garantire il contatto della zampa a terra.
Considerando il piano identificato si ha pertanto che la fase di stance si muove

nel piano P=(X,Y_{offset},Z_{offset}):

$$X = \frac{L}{2}t \tag{4.3}$$

con t parametro che varia tra -1 e 1.

L'unione della curva ellittica di swing e la retta della fase di stance,permettono di ottenere,per ciascuna zampa,una traiettoria complessiva del punto finale che è illustrata in figura 4.9.

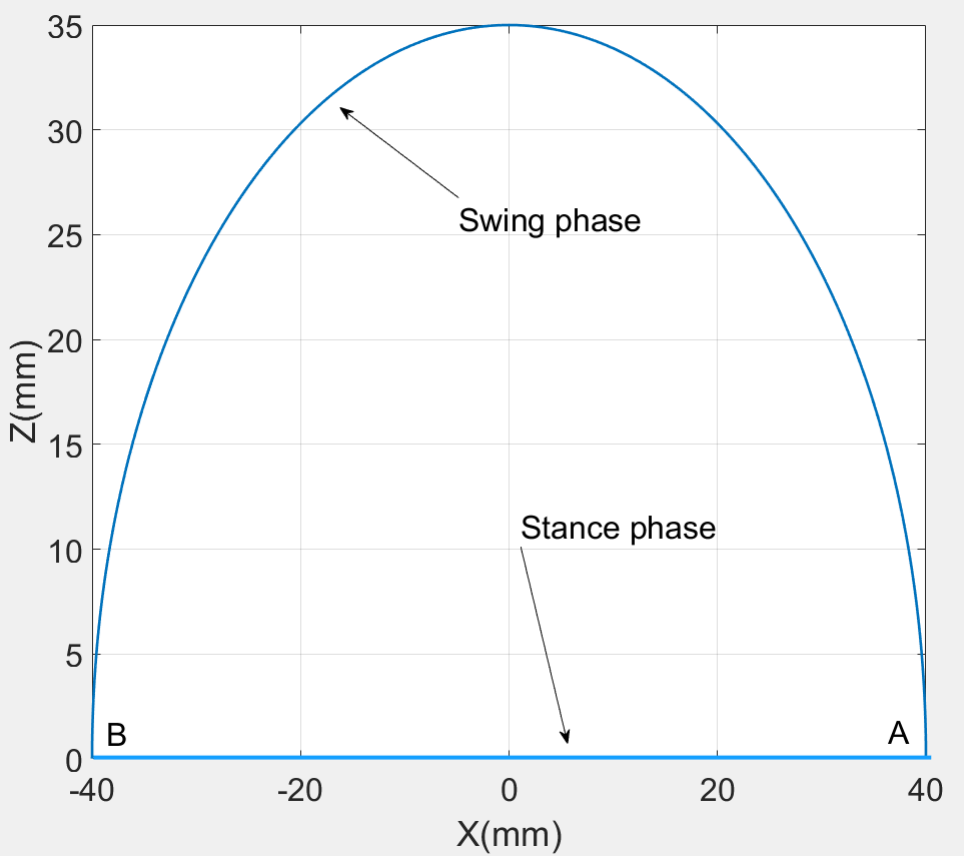

Figura 4.9: Forma della traiettoria nel piano XZ.

4.2 Controllo del movimento

La cinematica dei robot si occupa dello studio del movimento dei robot e delle loro componenti,concentrandosi principalmente sulla descrizione della posizione e dell'accelerazione dei link (gli elementi rigidi congiunti da articolazioni) che compongono un robot manipolatore,con l'obiettivo di posizionare la parte finale (end-effector) nella posizione voluta.

La cinematica dei robot può essere suddivisa in due tipologie principali: cinematica diretta e cinematica inversa.

4.2.1 Cinematica Diretta

La cinematica diretta ha come obbiettivo la determinazione del end-effector del braccio robotico rispetto agli angoli assunti dai giunti che lo costituiscono.Questo processo coinvolge la trasformazione delle coordinate dalle articolazioni alle coordinate del punto finale (end-effector) del robot.

4.2.2 Cinematica Inversa

La cinematica inversa consiste nella determinazione delle variabili articolari corrispondenti a una data posizione e orientamento del end-effector.La soluzione di questo problema è utile per trasformare le specifiche di movimento, assegnate al end-effector, nei corrispondenti movimenti (e quindi angoli) dei giunti, che consentono l'esecuzione del movimento desiderato.
Rispetto alla cinematica diretta,quella inversa risulta essere molto più complessa per una serie di ragioni tra le quali:

- Potrebbe non esserci alcuna soluzione ammissibile, a causa della struttura cinematica del manipolatore;
- Potrebbero esistere più soluzioni;
- Le equazioni da risolvere sono generalmente non lineari e quindi non sempre è possibile trovare una soluzione in forma chiusa.

Per ottenere una soluzione in forma chiusa al problema della cinematica inversa,esistono essenzialmente due tecniche:

- *Algebrica:* consiste in manipolazioni delle equazioni cinematiche fino ad ottenere un insieme di equazioni che permettano un'inversione delle equazioni;
- *Geometrica:* consiste su considerazioni di tipo geometrico,dipendenti dalla struttura del manipolatore. [11] .

4.3 Cinematica inversa applicata all'esapode

Per lo studio della cinematica inversa applicata al hexapod si è deciso di applicare un approccio di tipo geometrico,ricavando le equazioni dei tre angoli associati ai

tre giunti (coscia,femore,tibia) prendendo in esame una singola zampa e di adattare in seguito le equazioni ottenute alle restanti cinque.L'origine del sistema di riferimento è stato scelto nella posizione di collegamento fra la zampa e il corpo,con la posizione delle di riposo dell'esapode come riferimento degli assi,questo comporta che i valori delle coordinate x,y,z che vengono date per muovere l'intera struttura devono essere sommate ai valori di offset (Y_{offset} e Z_{offset}) che sono stati introdotti per avere nella condizione x=0,y=0,z=0 la posizione di riposo,come accennato nel Capitolo 4.1.2.
Considerando pertanto la zampa numero tre,si è deciso di orientare il sistema cartesiano come visibile in figura 4.10 e 4.11.

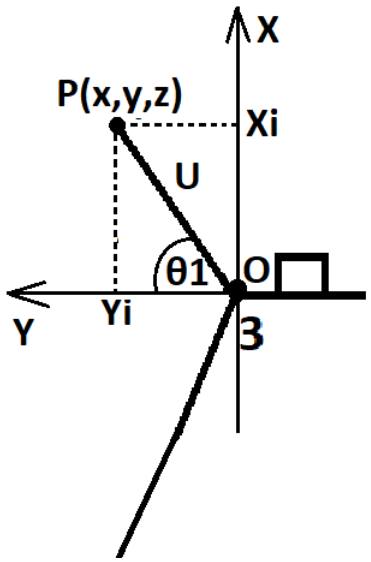

Figura 4.10: Zampa nel piano YX

In figura 4.10 è visibile la gamba numero tre, con il relativo sistema di riferimento nel piano YX.La zampa spostandosi in tale piano si muove con un angolo θ_1 rispetto all'asse Y.Per trattare la zampa nel movimento in altezza e in allungamento si è deciso di fissare un piano YZ.In questo,il punto P,che varia secondo le coordinate XYZ imposte per realizzare il movimento voluto dalla zampa, corrisponde al end-effector e si sposta in tale piano secondo le coordinate Xi,Yi,Zi.Il relativo piano YZ è visibile in figura 4.11. [12]

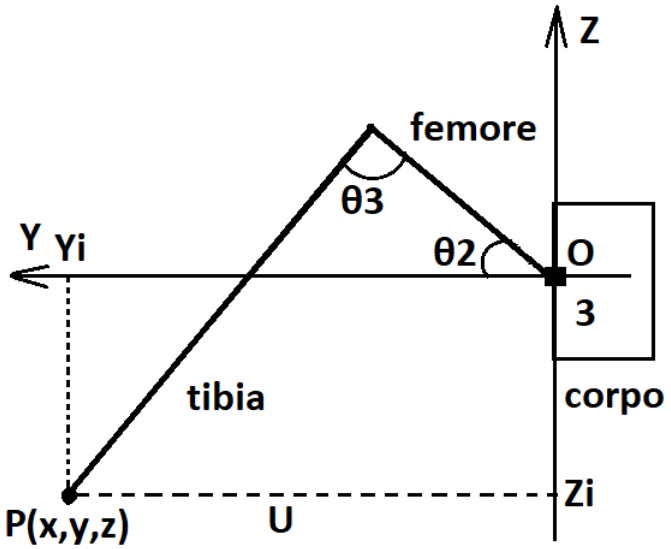

Figura 4.11: Zampa nel piano YZ

La zampa si sposta nel piano YX per muoversi verso la parte anteriore/posteriore del hexapod,mentre varia la sua altezza (angolo θ_2) e lunghezza (angolo θ_3) lungo il piano YZ.La lunghezza complessiva della zampa vista dall'alto è indicata con la lettera U.
Prendendo in esame la figura 4.10,è possibile ricavare l'angolo θ_1 conoscendo i valori di Xi e Yi:

$$\tan\theta_1 = \frac{Xi}{Yi} \tag{4.4}$$

Da cui è possibile ricavare il valore del angolo θ_1 che controlla il servo collegato alla Coscia:

$$\theta_1 = \arctan\left(\frac{Xi}{Yi}\right) \tag{4.5}$$

Risulta utile inoltre calcolare la lunghezza del segmento U,che verrà impiegata per successivi calcoli:

$$U = \sqrt{Xi^2 + Yi^2} \tag{4.6}$$

Considerando il piano YZ,si individua il triangolo rettangolo che si forma tra U,Zi,L(termine con cui viene indicata la distanza del punto P dall'origine O).

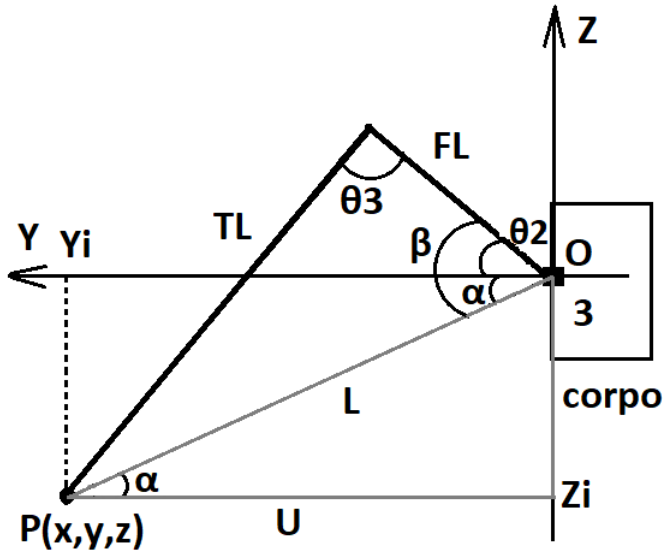

Figura 4.12: Zampa nel piano YZ.Con F_L si è indicata la lunghezza del femore,mentre con T_L quella della tibia

Il valore dell'angolo θ_2 può essere ottenuto da β+α,essendo il valore di α negativo.Questo perché a riposo, a causa dei valori di offset fissati,la coordinata Zi(che determina tale angolo) è negativa.
Come prima cosa viene calcolato il valore di L,necessario a trovare gli angoli β e α.Questo può essere ricavato grazie al teorema di Pitagora applicato al triangolo rettangolo di lati U,Zi,L:

$$L = \sqrt{Zi^2 + U^2} \tag{4.7}$$

Il valore di α può essere ottenuto considerando il triangolo costituito dai lati U,Zi,L.
In quanto triangolo rettangolo vale la formula $Zi = U\tan(\alpha)$,da cui si ottiene:

$$\alpha = \arctan\left(\frac{Zi}{U}\right) \tag{4.8}$$

Per il valore di β si può applicare il teorema del coseno al triangolo di lati F_L,T_L,L. Si ottiene perciò:

$$T_L^2 = F_L^2 + L^2 - 2LF_L\cos(\beta) \tag{4.9}$$

Da cui si ricava:

$$\beta = \arccos\left(\frac{F_L^2 + L^2 - T_L^2}{2LF_L}\right) \tag{4.10}$$

Pertanto il valore di θ_2,che controlla il servo associato al femore, risulta essere:

$$\theta_2 = \beta + \alpha = \arccos\left(\frac{F_L^2 + L^2 - T_L^2}{2LF_L}\right) + \arctan\left(\frac{Zi}{U}\right) \tag{4.11}$$

In ultima si calcola il valore dell'angolo θ_3 applicando il teorema del coseno al triangolo di lati F_L,T_L,L:

$$\theta_3 = \arccos\left(\frac{F_L^2 + T_L^2 - L^2}{2F_L T_L}\right) \tag{4.12}$$

Capitolo 5

Codice

5.1 Implementazione della Cinematica e della traiettoria

In questa sezione viene inizialmente presentata la matrice contenente i punti x,y,z necessari all'interpolazione della traiettoria che è implementata dall'unione delle due equazioni trovate nel capitolo 4.In seguito vengono illustrate e descritte le due funzioni più importanti del codice di Arduino utile a garantire il movimento dell'esapode:

- *newAngle:*funzione che permette di impiegare le equazioni della la cinematica inversa per calcolare gli angoli a partire dai valori di x,y,z dati;
- *Motion:*funzione che permette di svolgere il movimento della gamba implementando la traiettoria delle zampe e controllando il tripod gait necessario a coordinare le sei zampe.

5.1.1 Matrice trajectory

```
double trajectory[12][2] = {  //matrice per la traiettoria
  {0.0,00.0},//0
  {8.0,00.0},//1
  {20,00.0},//2
  {40,00.0},//3 corrisponde al massimo punto estremo destro A
 (40mm)
  {32,21.0},//4
  {20,30.3},//5
  {0.0,35.0},//6 corrisponde al massimo punto di altezza 35mm
```

```
    {-20.0,30.3},//7
    {-32.0,21.0},//8
    {-40.0,00.0},//9 corrisponde al massimo punto estremo sinistro B (-40mm)
    {-20.0,00.0},//10
    {-8.0,00.0},//11
  };

```

Listing 5.1: Codice della matrice trajectory

Per garantire una velocità maggiore al codice,nonché ridurne la complessità,si è deciso di impiegare la libreria Ramp.h fornita dal IDE di Arduino.Questa permette,fornendo il valore del punto da raggiungere, di creare una serie di valori in coordinate per giungere al punto finale voluto.La libreria offre diversi tipi di interpolazione, o in caso non espressamente definito,impiega quello che più si adatta a partire dai valori forniti.
Si è pertanto deciso di realizzare una matrice 12x2 definita trajectory contenete le coordinate x e z necessarie a realizzare la traiettoria.Tali punti sono stati scelti con l'impiego delle curve definite nelle equazioni 4.1,4.2,4.3.
Come deciso per la traiettoria voluta (visibile dalla figura 4.9),la matrice contiene i punti in coordinate espresse in millimetri, con valore massimo di altezza di 35mm e una fase di swing che va da -40mm a +40mm.Tali valori vanno a sommarsi ai valori di offset scelti per garantire che nella posizione x=0,y=0 e z=0 si abbia la posizione di riposo del end-effector.

5.1.2 Funzione newAngle

```
  void newAngle(double Xi, double Yi, double Zi, Giunto *Giunto1, Giunto *Giunto2, Giunto *Giunto3){
    // Applico offset Y e Z per la posizione di riposo
    Yi=Yi+Yoffset;
    Zi=Zi+Zoffset;
    //implementA_Zione delle formule ricavate
    //i valori degli angoli ottenuti mediante le funzioni atan, acos restituiscono
    //un valore in radianti che viene addatto in gradi
    double theta1=atan(Xi/Yi);
    double U=sqrt(pow(Yi,2) + pow(Xi,2));
    double L=sqrt(pow(U,2) + pow(Zi,2));
    double beta=acos((pow(L,2)+pow(FL,2)-pow(TL,2))/(2*L*FL))*(180/PI);
```

```
12     double alpha=atan(Zi/U)*(180/PI);
13     double theta2=(beta+alpha);  //il valore di alpha che si
    ottiene e' negativo percio' si somma
14     double theta3=acos((pow(FL,2)+pow(TL,2)-pow(L,2))/(2*FL*TL));
15     theta1= theta1*(180/PI);
16     theta3= theta3*(180/PI);
17     Giunto1->newValue((90 - theta1));
18     Giunto2->newValue(90 - theta2);
19     if (Giunto3->Inv){
20       Giunto3->newValue(theta3 - theta3offset);
21     }
22     else{
23       Giunto3->newValue(theta3 + theta3offset);
24     }
25    }
```

Listing 5.2: Codice della funzione newAngle

Nel pezzo di codice sopra riportato,che realizza la funzione newAngle,si sono implementate in Arduino le equazioni della cinematica inversa che sono state precedentemente ricavate dalle formule 4.5,4.11 e 4.12.
La funzione realizzata accetta in ingresso cinque parametri.Di questi i primi tre Xi,Yi e Zi, corrispondo alle coordinate a cui si deve posizionare l'end-effector,mentre i restanti tre sono degli oggetti definiti Giunti che rappresentano i servo che controllano rispettivamente la coscia,il femore e la tibia.Alla riga 3 e 4 sono applicati ai valori di Yi e Zi i valori di offset necessari a posizionare il sistema di riferimento con l'origine nella posizione di riposo dell'esapode.Alla riga 19 e 22 si è inserito un offset a causa della conformazione della tibia che non risulta effettivamente dritta ma leggermente curva.Dalla riga 8 alla riga 14 viene effettuato il calcolo della cinematica inversa con cui vengono determinati gli angoli necessari per i tre servo.Poiché le funzioni trigonometriche presenti nel IDE di Arduino restituiscono un valore in radianti,è stato necessario moltiplicare i risultati ottenuti per $\frac{180}{\pi}$ al fine di convertirli in gradi necessari ai servo.
In fine i risultati in gradi sono stati adattati all'intero range che può compiere un servomotore.Quest'ultimo infatti, si muove in angolo che va da 0° a 180°,mentre gli angoli calcolati vanno da -90° a +90°. Per far ciò si sono sottratti a 90° i valori degli angoli calcolati.Al termine sono stati assegnati gli angoli mediante la funzione Update realizzata per l'oggetto Giunto.

5.1.3 Funzione Motion()

```
void Motion(){
 if(stay){ //posizione di riposo
   StandUp();
 }
 else{
   // aggiornamento dei nuovi valori calcolati mediante
  interpolazione
   A_XPos = A_XEnd.update();
   A_YPos = A_YEnd.update();
   A_ZPos = A_ZEnd.update();
   A_XPos_turn=-A_XPos; //inverto i gradi per permettere di
  ottenere il verso opposto

   B_XPos = B_XEnd.update();
   B_YPos = B_YEnd.update();
   B_ZPos = B_ZEnd.update();
   B_XPos_turn=-B_XPos;
   if(turnRight | turnLeft){ //per girare e' necessario che 3
  gambe si muovano in verso opposto alle rimanenti
     L1.newAngle(A_XPos_turn,A_YPos,A_ZPos,&L1theta1,&L1theta2,&
  L1theta3);
     L2.newAngle(B_XPos_turn,B_YPos,B_ZPos,&L2theta1,&L2theta2,&
  L2theta3);
     L3.newAngle(A_XPos_turn,A_YPos,A_ZPos,&L3theta1,&L3theta2,&
  L3theta3);
     L4.newAngle(B_XPos,B_YPos,B_ZPos,&L4theta1,&L4theta2,&
  L4theta3);
     L5.newAngle(A_XPos,A_YPos,A_ZPos,&L5theta1,&L5theta2,&
  L5theta3);
     L6.newAngle(B_XPos,B_YPos,B_ZPos,&L6theta1,&L6theta2,&
  L6theta3);
   }
   else{
     L1.newAngle(A_XPos,A_YPos,A_ZPos,&L1theta1,&L1theta2,&
  L1theta3);
     L2.newAngle(B_XPos,B_YPos,B_ZPos,&L2theta1,&L2theta2,&
  L2theta3);
     L3.newAngle(A_XPos,A_YPos,A_ZPos,&L3theta1,&L3theta2,&
  L3theta3);
     L4.newAngle(B_XPos,B_YPos,B_ZPos,&L4theta1,&L4theta2,&
  L4theta3);
     L5.newAngle(A_XPos,A_YPos,A_ZPos,&L5theta1,&L5theta2,&
  L5theta3);
     L6.newAngle(B_XPos,B_YPos,B_ZPos,&L6theta1,&L6theta2,&
```

```
L6theta3);
 }


 // alla fine del calcolo di un singolo step di interpolazione viene eseguito il successivo
 if ((Go==false | A_XEnd.isFinished()==true)){
   if(forward | turnLeft){ //per eseguire il movimento in avanti e girare a sinistra si incrementano gli step che potrano scorrere la matrice definita trajectory
     commandStepA++;
     commandStepB++;
     if (commandStepA> 11){
       commandStepA = 0;
     }
     if (commandStepB > 11){
       commandStepB = 0;
     }
   }
   if(backward | turnRight){//per eseguire il movimento all' indietro e girare a destra si decrementano gli step che potrano scorrere la matrice definita trajectory
     commandStepA--;
     commandStepB--;
     if (commandStepA<=0){
       commandStepA = 11;
     }
     if (commandStepB<=0){
       commandStepB = 11;
     }
   }
   xMov = trajectory[commandStepA][0];
   yMov=0; //per la traiettoria ci si sposta lungo il piano XZ mentre il valore di y rimane nullo,rimane percio' il valore di offset fissato
   zMov = trajectory[commandStepA][1];
   int duration = trajectorySpeed;
   A_XEnd.go(xMov,duration);    //funzione che calcola l' interpolazione per il nuovo valore con il tempo per il movimento
   A_YEnd.go(yMov,duration);
   A_ZEnd.go(zMov,duration);
   xMov = trajectory[commandStepB][0];
   zMov = trajectory[commandStepB][1];
```

```
65        duration = trajectorySpeed;
66        B_XEnd.go(xMov,duration);
67        B_YEnd.go(yMov,duration);
68        B_ZEnd.go(zMov,duration);
69        Go =true;
70      }
71    }
72  }
```

Listing 5.3: Codice della funzione Motion()

La seguente funzione permette di eseguire la traiettoria del movimento mediante l'interpolazione dai valori X e Z definiti nella matrice trajectory. Per utilizzare la libreria Ramp.h sono stati creati sei oggetti di tipo rampDouble, tre per il gruppo A (A_XEnd,A_YEnd,A_ZEnd) e altrettanti per il gruppo B (B_XEnd,B_YEnd,B_ZEnd) che corrispondono alle coordinate x,y,z di ciascun gruppo di zampe. Sono state create altrettante variabili del tipo A_XPos che salvano il valore calcolato per mezzo degli oggetti precedentemente definiti e le passano alla funzione newAngle. Nella prima parte del codice vengono aggiornati i valori delle successive coordinate determinati mediante interpolazione.Per fare ciò viene impiegata la funzione update() resa disponibile dalla libreria Ramp.h.
I nuovi valori di x,y,z vengono passati alla funzione newAngle che trova gli angoli necessari a posizionare l'end-effector nella nuova posizione.Per realizzare la rotazione a destra e a sinistra,come spiegato nel capitolo 4, è necessario che le zampe di un lato si muovano in direzione opposta a quelle dell'altro.Per fare ciò si determina il valore opposto per mezzo delle variabili A_XPos_turn e A_XPos_turn. Alla riga 35 e 46 sono presenti due if che vengono eseguiti a seconda del movimento voluto.In caso di movimento in avanti e di rotazione a sinistra il primo ciclo determina l'incremento degli step necessari per eseguire il movimento del gruppo A e del gruppo B. Al contrario se viene eseguito il movimento all'indietro o la rotazione a destra gli step vengono decrementati.
In ultima vengono assegnati i valori presenti nella matrice trajectory alla funzione go() che permette di fissare le coordinate dei punti di arrivo e il tempo necessario per farlo.

5.2 Controllo bluetooth

```
1 while (Serial3.available()){
```

```
2  com=char(Serial3.read()); // "com" e' il comando che viene
    ricevuto
3  switch(com){
4    case 'F': //in caso si riceva il comando F viene attivata la
    modalita'  movimento in avanti
5    {
6      stay=false;
7      backward = false;
8      forward = true;
9      turnRight = false;
10     turnLeft = false;
11     Motion();
12     break;
13   }
14   case 'B': //in caso si riceva il comando B viene attivata la
    modalita' movimento all'indietro
15   {
16     stay=false;
17     backward = true;
18     forward = false;
19     turnRight = false;
20     turnLeft = false;
21     Motion();
22     break;
23   }
24   case 'L': //in caso si riceva il comando L viene attivata la
    rotazione verso sinistra
25   {
26     stay=false;
27     backward = false;
28     forward = false;
29     turnRight = false;
30     turnLeft = true;
31     Motion();
32     break;
33   }
34   case 'R': //in caso si riceva il comando R viene attivata la
    rotazione verso destra
35   {
36     stay=false;
37     backward = false;
38     forward = false;
39     turnRight = true;
40     turnLeft = false;
```

```
41        Motion();
42        break;
43      }
44      case 'X':   //attivazione modalita' automatica e accensione
    led stato modalita' automatica(rosa)
45      {
46        autom=true;
47        digitalWrite(pinR_State,255);
48        digitalWrite(pinG_State,0);
49        digitalWrite(pinB_State,255);
50        break;
51      }
52      case 'x':   //disattivazione modalita' automatica
53      {
54        autom=false;
55        break;
56      }
57    }
58
```

Listing 5.4: Codice controllo bluetooth

La parte di codice sopra riportata ha il compito di ricevere i comandi inviati tramite bluetooth dal cellulare e di attuare i movimenti voluti.Per inviare i comandi dal cellulare si è fatto uso di un'applicazione già realizzata.
Nel ciclo while vengono letti dalla porta Serial3,che in Arduino Mega corrisponde ai pin 14 e 15,i valori ricevuti dal modulo bluetooth.I valori sono lettere associate alle direzioni che sono state scelte con la pressione dei corrispondenti pulsanti sul cellulare.
Ad ogni lettera vengono associati dei movimenti,con cui grazie ad un switch-case vengono attivate le variabili che identificano il movimento voluto.Con la modalità autonoma, identificata dalla lettera X, viene anche attivata la colorazione viola del led rgb per segnalare tale stato.Questa modalità viene invece disattiva con la lettera x, che permette anche di variare la colorazione del led che diviene blu segnalando il controllo in modalità manuale del robot.

5.3 Funzionamento autonomo

```
1 void AutoMotion(){
2   int back=0;
```

```
3   Distance(); //questa funzione calcola la distanza per mezzo del
      sensore ad ultrasuoni
4   if((distance<=30)&&(distance!=0)){ //controllo se si trova un
     ostacolo a distanza inferiore a 30cm
5     i=0;
6     back=0;
7     while((i<2)&&(back==0)){ //in presenza di ostacolo viene
     effettuato un movimento all'indietro di 2 passi
8       backward = true;
9       forward = false;
10      turnRight = false;
11      turnLeft = false;
12      Motion();
13      curMillis=millis();
14      if(curMillis-prevMillis_back>=interv){
15        prevMillis_back=curMillis;
16        i++;
17      }
18    }
19    back=1;
20    if(back==1){
21      i=0;
22    }
23    //  prevMillis_back=0;
24    while((i<3)){ //successivamente viene ruotato l'esapode verso
      destra compiendo una rotazione di 90 gradi che richiede 3
     passi
25      backward = false;
26      forward = false;
27      turnRight = true;
28      turnLeft = false;
29      Motion();
30      curMillis=millis();
31      if(curMillis-prevMillis_turn>=interv){
32        prevMillis_turn=curMillis;
33        i++;
34      }
35    }
36  }
37  else{ //in caso non si trovi ostacolo si procede con un
     movimento in avanti
38    stay=false;
39    backward = false;
40    forward = true;
```

```
41      turnRight = false;
42      turnLeft = false;
43      Motion();
44    }
45  }
```

Listing 5.5: Codice funzione AutoMotion()

La funzione AutoMotion(),ha il compito di gestire il controllo dell'esapode in assenza di comandi forniti dall'utente.Per fare questo viene impiegato il sensore ad ultrasuoni che, mediante la funzione realizzata ovvero Distance(), misura la distanza della struttura da eventuali oggetti posti davanti.
In caso sia presente un ostacolo a una distanza inferiore ai 30cm viene eseguita inizialmente un movimento all'indietro di due passi che consentono di eseguire la rotazione della struttura verso destra di 90° senza che questa vada ad urtare l'ostacolo posto di fronte.Il movimento all'indietro viene eseguito per due passi che impiegano ciascuno un intervallo di 1s,mentre la rotazione di 90° prevede 3 passi eseguiti ciascuno nello stesso intervallo di tempo.
In assenza di ostacoli l'esapode procede in avanti.

5.4 Monitoraggio carica batteria Lipo

```
1  void Battery(){
2    value = analogRead(A8); //lettura del valore analogico della
      batteria
3    voltage = value*(5.0/1023.0)*3.0; //conversione del valore in
      una scala che giunga a 12.6v
4    if(voltage>12){ //stato di carica completa con accensione led
      rgb colore verde
5      digitalWrite(pinR_Bat,0);
6      digitalWrite(pinG_Bat,255);
7      digitalWrite(pinB_Bat,0);
8    }
9    else{
10     if(voltage>=11.5){ //stato di carica media con accensione led
       rgb colore giallo
11       digitalWrite(pinR_Bat,255);
12       digitalWrite(pinG_Bat,255);
13       digitalWrite(pinB_Bat,0);
14     }
15     else{ //stato di carica basso con accensione led rgb colore
      rosso
```

```
16        digitalWrite(pinR_Bat,255);
17        digitalWrite(pinG_Bat,0);
18        digitalWrite(pinB_Bat,0);
19      }
20    }
21  }
```

Listing 5.6: Codice funzione Battery()

Quest'ultima funzione monitora lo stato di carica della batteria Lipo impiegata.Il valore di tensione della batteria,che al massimo stato di carica presenta una tensione di 12.6V, viene ridotto attraverso un partitore,ottenendo un valore massimo al pin A8 di 4.2V.
Dopo aver letto il valore di tensione dal pin,alla riga 3 viene riscalato il valore letto a una scala che associ ai 4.2V letti il valore originale della batteria di 12.6V. In seguito a seconda del valore di tensione della batteria vengono assegnati tre distinti colori:

- *Verde:* identifica la batteria carica con un valore di tensione superiore ai 12V;
- *Giallo:* identifica un livello di carica medio con un valore di tensione della batteria compreso tra 12V e 11.5V;
- *Rosso:* identifica un livello di carica basso con un valore di tensione della batteria inferiore a 11.5V.

Capitolo 6

Verifica del funzionamento

In questo capitolo vengono analizzate eventuali problematiche riscontrate nella fase di verifica del funzionamento dell'esapode.
Dopo un'iniziale assemblaggio della struttura ci si è accorti che, a seconda della posizione di fissaggio dei 6 servo della coscia (ovvero l'insieme dei servo denominati Giunto1) al corpo,si aveva un influenza sulla massima escursione che questi potevano compiere.In particolare si è notato che tali servo, per come fissati,potevano eseguire solo metà del raggio di movimento voluto spostando la zampa solo all'indietro e raggiungendo la massima posizione (180°) nella posizione di riposo,non riuscendo perciò a compiere il movimento in avanti.
Per risolvere questo problema è stato necessario smontare l'intera struttura e collegare tutti i 6 servo ad Arduino e programmarli affinché si posizionassero tutti a 90°.In seguito mantenendoli alimentati,per garantire che rimanessero nell'angolazione fissata,sono stati fissati nella posizione di riposo (vedere figura 4.2) rimontando l'intera struttura.Questo approccio ha permesso di avere un pari range di movimento in entrambe le direzioni essendo il servo in grado di assumere posizioni nel range compreso tra 0° e 180°.
Durante la fase di collaudo della posizione di riposo si è notato che i vari servo si posizionavano in modo differente con un commando di 0° a causa della diversa conformazione della struttura.Ciò portava le zampe ad assumere posizioni differenti in altezza e in larghezza(ovvero avevano posizioni differenti sul piano YZ).Per rimediare a ciò si è provveduto via software a inserire degli offset a tutti i 18 servo con l'obiettivo che nella posizione di riposo si avesse la stessa posizione per ogni zampa.Si è provveduto perciò ad appoggiare l'intera struttura su un supporto rialzato posto sopra ad un tavolo e si sono inseriti degli offset con l'obiettivo che le varie zampe si posizionassero ad un'altezza dal tavolo che fos-

se uguale per tutte e,aggiustando conseguentemente i vari valori.In seguito si è provveduto ad effettuare,con la stessa metodologia,la regolazione dei servo che controllano l'angolo θ_3 con l'obiettivo che tutte e sei le zampe avessero la tibia in una posizione tale da garantire un valore di U che fosse uguale per ogni zampa.
Durante il collaudo del pezzo di codice inerente al calcolo della cinematica inversa,si è notato alcune imprecisioni fra i gradi calcolati e quelli a cui effettivamente si posizionavano i 6 servomotori che controllavano il Giunto3,o analogamente l'angolo θ_3.Questo è dovuto alla diversa conformazione della tibia che non è perfettamente dritta come una retta ma si presenta leggermente curva (vedere figura 3.2).Questo ha portato in alcuni casi a dover inserire un offset (theta3offset) nella fase di calcolo della cinematica inversa come spiegato nel capitolo 5.1.2.

Capitolo 7

Possibili migliorie

Al termine di tale progetto si sono riscontrate alcune migliorie soprattutto in fase di collaudo dell'esapode.Di seguito ne sono elencate alcune che potrebbero essere applicate al seguente progetto:

- *Servo:*il modello di servo impiegato in questo progetto risulta avere qualche limitazione sia nella massima coppia di stallo sia nella precisione nell'assumere le posizione volute.Questo problema si potrebbe superare con l'impiego di servomotori di fascia più alta che garantiscano coppie di stallo maggiori e maggiore precisione nell'assumere le angolazioni volute, garantendo un movimento più fluido della locomozione;

- *Microcontrollore:*l'impiego di un dispositivo come Raspberry Pi rispetto ad Arduino migliorerebbe notevolmente il controllo dell'intero esapode.Raspberry, rispetto ad Arduino,è un computer a tutti gli effetti.Ciò significa che presenta una maggiore potenza di calcolo e può gestire compiti più complessi,come il riconoscimento di immagini o l'esecuzione di algoritmi di intelligenza artificiale per il controllo avanzato,oltre a disporre di moduli Bluetooth e Wi-Fi già integrati.

- *Sensori:*l'utilizzo di ulteriori sistemi di rilevamento degli ostacoli possono ridurre eventuali urti delle zampe contro oggetti posti lungo i lati del robot che possono non essere rilevati dal sensori ad ultrasuoni che è posizionato fisso nella parte anteriore.Alcuni esempi possono essere l'inserimento di ulteriori due sensori ad ultrasuoni fissi ai lati dell'esapode o in alternativa di una telecamera fissa nella parte anteriore che possa ruotare di 180°;

- *Comunicazione:*la possibile integrazione di un controllo wi-fi,che permetta al robot di poter avere un raggio di copertura molto più ampio di quella offerta

con una semplice connessione bluetooth.Questo consentirebbe il controllo anche da grandi distanze.

- *Funzionalità aggiuntive:*la possibile implementazione di tecniche di apprendimento automatico e d'intelligenza artificiale per consentire al robot di apprendere e migliorare la sua capacità di adattarsi nel tempo al terreno su cui viene impiegato.Ciò potrebbe aiutare il robot a riconoscere pattern nel terreno e a sviluppare strategie di movimento più efficienti.Questo consentirebbe al hexapod di cambiare modalità di locomozione in base alle condizioni del terreno con una regolazione dinamica della posizione e dell'orientamento delle zampe in risposta alle variazioni del terreno.

Capitolo 8

Conclusioni

Il progetto trattato analizza le fasi di realizzazione di un robot esapode con relativo controllo a distanza per mezzo di uno smartphone con l'impiego di un modulo Bluetooth.Presenta in alternativa un controllo autonomo da parte dell'esapode stesso.Quest'ultimo gli permette di muoversi nell'ambiente circostante senza la necessità di supervisione grazie all'impiego di un sensore ad ultrasuoni per la rilevazione di eventuali ostacoli che possano essere in tal modo evitati.Foto del progetto concluso sono visibili in appendice B.2

Per garantire un progetto affidabile dal punto di vista delle prestazione in una prima fase si è ricercato il materiale necessario a realizzare l'intero robot.Non avendo gli strumenti necessari,e con l'obiettivo di ridurre il tempo impiegato nella realizzazione,si è optato per acquistare una struttura metallica già realizzata.Dopo di che si è provveduto a scegliere l'hardware necessario,come servomotori e scheda di controllo come discusso nel Capitolo 3.Dopo aver stabilito i componenti necessari all'attuazione del progetto,si è provveduto alla realizzazione dello schema del circuito con conseguente progettazione del PCB necessario,visibile in Appendice B.1.

In un secondo momento si è affrontato invece lo studio teorico alla base del movimento. Per poter garantire una locomozione adeguata dell'intera struttura,con annessa coordinazione delle sei zampe,è stato necessario concentrarsi sulla teoria che sta alla base del movimento delle zampe suddividendo questa in due fasi.

Nello specifico in una prima fase è stato studiando l'aspetto inerente allo schema con cui le sei zampe avessero dovuto muoversi per poter garantire da un lato una locomozione stabile ma che dall'altro ciò non precludesse una velocità adeguata dei movimenti.Per tale motivo come spiegato nel capitolo 4.1 si è optato per uno schema di movimento basato su tripod gait studiando successivamente come le

sei zampe avessero dovuto muoversi per garantire i movimenti voluti.
Nella seconda fase si è provveduto invece allo studio della cinematica inversa necessaria,come illustrato nel Capitolo 4.3,a ricavare gli angoli dei vari servomotori in funzione delle coordinate x,y,z fornite al hexapod.
Tutto ciò ha portato alle conoscenze adeguate per poter realizzare il codice Arduino necessario a garantire il funzionamento voluto.
Di questo i principali aspetti sono stati presentati e descritti nel Capitolo 5.
Il codice si basa inizialmente sulla scelta da parte di un operatore del tipo di movimento voluto,in base a cui vengono stabilite le coordinate x,y,z(presenti nella matrice trajectory) necessarie a compierlo.Tali coordinate vengono in seguito utilizzate,per mezzo delle funzioni messe a disposizione della libreria Ramp.h,per l'interpolazione necessaria a formare la traiettoria che le zampe devono compiere.Le coordinate ricavate nel piano XYZ vengono in seguito impiegate per il calcolo della cinematica inversa necessaria a stabilire gli angoli che i vari servono devono assumere,ciò viene effettuato per mezzo della funzione NewAngle.L'intero codice realizzato è consultabile in Appendice A.
In conclusione il progetto può prevedere alcuni futuri miglioramenti come illustrato nel Capitolo 7.Di questi la possibile integrazione di Raspberry al posto di Arduino porterebbe alla possibilità di impiegare algoritmi d'intelligenza artificiale per il controllo.

Appendice A

Codice Arduino

```
#include <Servo.h>
#include <Math.h>
#include <Ramp.h> //libreria per interpolazione
#define trig 11 //pin sensore ad ultrasuoni
#define echo 12
#define pinR_State 5 //pin led RGB controllo automatico/manuale
#define pinG_State  6
#define pinB_State  7
#define pinR_Bat 10 //pin led RGB segnalazione batteria
#define pinG_Bat  9
#define pinB_Bat  8
int value;
float voltage;
double curMillis=0;
double prevMillis=0;
double prevMillis_bat=0;
double prevMillis_back=0;
double prevMillis_turn=0;
int interv=1000;
float duration = 0;
float distance = 0;
bool backward = false;
bool forward = false;
bool turnRight = false;
bool turnLeft = false;
bool stay = true;
char com;
bool autom=false;
int i=0;
```

```
30      double FL = 90.0; // lunghezza del femore
31      double TL = 120.0; // lunghezza della tibia
32      double xMov=0;
33      double yMov=0;
34      double zMov=0;
35      double Yoffset =  80.0; //offset decisi in base alla
    posizione di riposo voluta
36      double Zoffset = -130.0;
37      double theta3offset = 10;
38
39       double trajectory[12][2] = {  //matrice per la traiettoria
40       {0.0,00.0},//0
41       {8.0,00.0},//1
42       {20,00.0},//2
43       {40,00.0},//3 corrisponde al massimo punto estremo destro A
     (40mm)
44       {32,21.0},//4
45       {20,30.3},//5
46       {0.0,35.0},//6 corrisponde al massimo punto di altezza 35mm
47       {-20.0,30.3},//7
48       {-32.0,21.0},//8
49       {-40.0,00.0},//9 corrisponde al massimo punto estremo
    sinistro B (-40mm)
50       {-20.0,00.0},//10
51       {-8.0,00.0},//11
52     };
53
54      //definzione delle variabili necessarie alla funzione Motion
    ()
55      bool Go = false;
56      int commandStepA = 11;
57      int commandStepB = 5;
58      // Variabili necessarie per la funzione Motion()
59      double A_XPos = 0.0;
60      double A_XPos_turn = 0.0;
61      double A_YPos = 0.0;
62      double A_ZPos = 0.0;
63
64      double B_XPos = 0.0;
65      double B_XPos_turn = 0.0;
66      double B_YPos = 0.0;
67      double B_ZPos = 0.0;
68      //oggetti di tipo rampDouble dalla libreria Ramp
69      rampDouble A_XEnd = 0.0;
```

```
70      rampDouble A_YEnd = 0.0;
71      rampDouble A_ZEnd = 0.0;
72
73      rampDouble B_XEnd = 0.0;
74      rampDouble B_YEnd = 0.0;
75      rampDouble B_ZEnd = 0.0;
76
77      //definizione pin dei vari servo associati ai giunti
78      #define L1theta1Pin 40
79      #define L1theta2Pin 38
80      #define L1theta3Pin 36
81
82      #define L2theta1Pin 42
83      #define L2theta2Pin 44
84      #define L2theta3Pin 46
85
86      #define L3theta1Pin 48
87      #define L3theta2Pin 49
88      #define L3theta3Pin 47
89
90      #define L4theta1Pin 28
91      #define L4theta2Pin 30
92      #define L4theta3Pin 32
93
94      #define L5theta1Pin 24
95      #define L5theta2Pin 25
96      #define L5theta3Pin 26
97
98      #define L6theta1Pin 34
99      #define L6theta2Pin 23
100     #define L6theta3Pin 22
101
102     //classe Giunto
103     class Giunto {
104      public:
105      Servo ServoX;
106      bool Inv;
107      int AngleOffset;
108      int PinX;
109      Giunto(){}
110     //Metodo necessario a inizializzare i servo con relativo
    offset
111     void Setup(int Pin, bool I, int Angleoff){
112       PinX = Pin;
```

```
113         Inv = I;
114         AngleOffset = Angleoff;
115         ServoX.attach(PinX, 500, 2500);
116       }
117
118       void newValue(double Angle){
119         if (Inv){
120           ServoX.write(180 - Angle + AngleOffset);
121         }
122         else{
123           ServoX.write(Angle + AngleOffset);
124         }
125       }
126     };
127
128     //classe Leg realizzata per associare a ogni zampa i valori
    dei tre servo
129     class Leg {
130      public:
131      Leg(){
132      }
133       void newAngle(double Xi, double Yi, double Zi, Giunto *
    Giunto1, Giunto *Giunto2, Giunto *Giunto3){
134         // Applico offset Y e Z per la posizione di riposo
135         Yi=Yi+Yoffset;
136         Zi=Zi+Zoffset;
137         //implementazione delle formule ricavate
138         //i valori degli angoli ottenuti mediante le funzioni
    atan,acos restituiscono
139         //un valore in radianti che viene portato in gradi
140         double theta1=atan(Xi/Yi);
141         double U=sqrt(pow(Yi,2) + pow(Xi,2));
142         double L=sqrt(pow(U,2) + pow(Zi,2));
143         double beta=acos((pow(L,2)+pow(FL,2)-pow(TL,2))/(2*L*FL))
    * (180/PI);
144         double alpha=atan(Zi/U)*(180/PI);
145         double theta2=(beta+alpha);  //il valore di alpha che si
    ottiene e' negativo percio' si somma
146         double theta3=acos((pow(FL,2)+pow(TL,2)-pow(L,2))/(2*FL*
    TL));
147         theta1= theta1*(180/PI);
148         theta3= theta3*(180/PI);
149         Giunto1->newValue((90 - theta1));
150         Giunto2->newValue(90 - theta2);
```

```
151         if (Giunto3->Inv){
152           Giunto3->newValue(theta3 - theta3offset);
153         }
154         else{
155           Giunto3->newValue(theta3 + theta3offset);
156         }
157       }
158     };
159
160     // definizione di sei oggetti Leg associate alle sei zampe
161     Leg L1;
162     Leg L2;
163     Leg L3;
164     Leg L4;
165     Leg L5;
166     Leg L6;
167
168     //definizione dei 6 giunti
169     Giunto L1theta1; //zampa 1
170     Giunto L1theta2;
171     Giunto L1theta3;
172
173     Giunto L2theta1; //zampa 2
174     Giunto L2theta2;
175     Giunto L2theta3;
176
177     Giunto L3theta1; //zampa 3
178     Giunto L3theta2;
179     Giunto L3theta3;
180
181     Giunto L4theta1; //zampa 4
182     Giunto L4theta2;
183     Giunto L4theta3;
184
185     Giunto L5theta1; //zampa 5
186     Giunto L5theta2;
187     Giunto L5theta3;
188
189     Giunto L6theta1; //zampa 6
190     Giunto L6theta2;
191     Giunto L6theta3;
192
193     void StandUp(){ //funzione che definisce gli angoli per la
    posizione di riposo
```

```
194
195      L1.newAngle(0,0,0,&L1theta1,&L1theta2,&L1theta3);
196      L2.newAngle(0,0,0,&L2theta1,&L2theta2,&L2theta3);
197      L3.newAngle(0,0,0,&L3theta1,&L3theta2,&L3theta3);
198      L4.newAngle(0,0,0,&L4theta1,&L4theta2,&L4theta3);
199      L5.newAngle(0,0,0,&L5theta1,&L5theta2,&L5theta3);
200      L6.newAngle(0,0,0,&L6theta1,&L6theta2,&L6theta3);
201     }
202
203     int trajectorySpeed=100; // velocita' con cui viene eseguita
    l'interpolazione
204
205     void setup() {
206      pinMode(pinR_State, OUTPUT);
207      pinMode(pinG_State, OUTPUT);
208      pinMode(pinB_State, OUTPUT);
209      pinMode(pinR_Bat, OUTPUT);
210      pinMode(pinG_Bat, OUTPUT);
211      pinMode(pinB_Bat, OUTPUT);
212      Serial3.begin(9600); //setto baud rate per bluetooth
213      pinMode(trig, OUTPUT);
214      pinMode(echo, INPUT);
215
216
217      //calibrazione dei vari servo per permettere alle sei zampe
    di avere la stessa posizione
218      L1theta1.Setup(L1theta1Pin,false,-40);
219      L1theta2.Setup(L1theta2Pin,true, -1);
220      L1theta3.Setup(L1theta3Pin,false,-30);
221
222      L2theta1.Setup(L2theta1Pin,false,-25);
223      L2theta2.Setup(L2theta2Pin,true,-3);
224      L2theta3.Setup(L2theta3Pin,false,-5);
225
226      L3theta1.Setup(L3theta1Pin,false,22);
227      L3theta2.Setup(L3theta2Pin,true,7);
228      L3theta3.Setup(L3theta3Pin,false,-65);
229
230      L4theta1.Setup(L4theta1Pin,true,-32);
231      L4theta2.Setup(L4theta2Pin,false,-10);
232      L4theta3.Setup(L4theta3Pin,true,-17);
233
234      L5theta1.Setup(L5theta1Pin,true,25);
235      L5theta2.Setup(L5theta2Pin,false,-7);
```

```
236       L5theta3.Setup(L5theta3Pin,true,-20);
237
238       L6theta1.Setup(L6theta1Pin,true,-50);
239       L6theta2.Setup(L6theta2Pin,false,-10);
240       L6theta3.Setup(L6theta3Pin,true,-5);
241
242       // Stand Up
243       StandUp();
244       delay(2000);
245     }
246
247     void loop() {
248       curMillis=millis();
249       if(curMillis-prevMillis_bat>=interv){
250         prevMillis_bat=curMillis;
251         Battery(); //controllo stato batteria
252       }
253       while (Serial3.available()){
254         com=char(Serial3.read()); // "com" e' il comando che
    viene ricevuto
255         switch(com){
256           case 'F': //in caso si riceva il comando F viene
    attivata la modalita' movimento in avanti
257           {
258             stay=false;
259             backward = false;
260             forward = true;
261             turnRight = false;
262             turnLeft = false;
263             Motion();
264             break;
265           }
266           case 'B': //in caso si riceva il comando B viene
    attivata la modalita' movimento all'indietro
267           {
268             stay=false;
269             backward = true;
270             forward = false;
271             turnRight = false;
272             turnLeft = false;
273             Motion();
274             break;
275           }
```

```
276         case 'L': //in caso si riceva il comando L viene
    attivata la rotazione verso sinistra
277         {
278           stay=false;
279           backward = false;
280           forward = false;
281           turnRight = false;
282           turnLeft = true;
283           Motion();
284           break;
285         }
286         case 'R': //in caso si riceva il comando R viene
    attivata la rotazione verso destra
287         {
288           stay=false;
289           backward = false;
290           forward = false;
291           turnRight = true;
292           turnLeft = false;
293           Motion();
294           break;
295         }
296         case 'X':   //attivazione modalita' automatica e
    accensione led stato modalita' automatica(viola)
297         {
298           autom=true;
299           digitalWrite(pinR_State,255);
300           digitalWrite(pinG_State,0);
301           digitalWrite(pinB_State,255);
302           break;
303         }
304         case 'x':   //disattivazione modalita' automatica
305         {
306           autom=false;
307           break;
308         }
309       }
310       if(autom){
311         AutoMotion();
312       }
313       else{
314         digitalWrite(pinR_State,LOW);
315         digitalWrite(pinG_State,LOW);
316         digitalWrite(pinB_State,HIGH);
```

```
317         }
318         curMillis=millis();
319         if(curMillis-prevMillis_bat>=interv){
320           prevMillis_bat=curMillis;
321           Battery();
322         }
323       }
324 
325     }
326     void Distance(){ //funzione che permette di misurare la
    distanza da eventuali oggetti posti di fronte
327       digitalWrite(trig,LOW);
328       delayMicroseconds(3);
329       digitalWrite(trig,HIGH);
330       delayMicroseconds(10);
331       digitalWrite(trig,LOW);
332       curMillis=millis();
333       if((curMillis-prevMillis>=500)||distance==0){//ogni 0.5s
    viene rilevata una nuova distanza
334         prevMillis=curMillis;
335         duration=pulseIn(echo,HIGH);
336         distance = duration/58;
337       }
338     }
339     void AutoMotion(){
340       int back=0;
341       Distance(); //questa funzione calcola la distanza per mezzo
     del sensore ad ultrasuoni
342       if((distance<=30)&&(distance!=0)){ //controllo se si trova
    un ostacolo a distanza inferiore a 30cm
343         i=0;
344         back=0;
345         while((i<2)&&(back==0)){ //in presenza di ostacolo viene
    effettuato un movimento all'indietro di 2 passi
346           backward = true;
347           forward = false;
348           turnRight = false;
349           turnLeft = false;
350           Motion();
351           curMillis=millis();
352           if(curMillis-prevMillis_back>=interv){
353             prevMillis_back=curMillis;
354             i++;
355           }
```

```
356       }
357       back=1;
358       if(back==1){
359         i=0;
360       }
361       //  prevMillis_back=0;
362       while((i<3)){ //successivamente viene ruotato l'esapode
  verso destra compiendo una rotazione di 90 gradi che richiede
  3 passi
363         backward = false;
364         forward = false;
365         turnRight = true;
366         turnLeft = false;
367         Motion();
368         curMillis=millis();
369         if(curMillis-prevMillis_turn>=interv){
370           prevMillis_turn=curMillis;
371           i++;
372         }
373       }
374     }
375     else{ //in caso non si trovi ostacolo si procede con un
  movimento in avanti
376       stay=false;
377       backward = false;
378       forward = true;
379       turnRight = false;
380       turnLeft = false;
381       Motion();
382     }
383   }
384   void Battery(){
385     value = analogRead(A8); //lettura del valore analogico
  della batteria
386     voltage = value*(5.0/1023.0)*3.0; //conversione del valore
  in una scala che giunga a 12.6v
387     if(voltage>12){ //stato di carica completa con accensione
  led rgb colore verde
388       digitalWrite(pinR_Bat,0);
389       digitalWrite(pinG_Bat,255);
390       digitalWrite(pinB_Bat,0);
391     }
392     else{
```

```
393         if(voltage>=11.5){ //stato di carica media con accensione
     led rgb colore giallo
394           digitalWrite(pinR_Bat,255);
395           digitalWrite(pinG_Bat,255);
396           digitalWrite(pinB_Bat,0);
397         }
398         else{ //stato di carica basso con accensione led rgb
    colore rosso
399           digitalWrite(pinR_Bat,255);
400           digitalWrite(pinG_Bat,0);
401           digitalWrite(pinB_Bat,0);
402         }
403       }
404      }
405
406      void Motion(){
407       if(stay){ //posizione di riposo
408         StandUp();
409       }
410       else{
411         // aggiornamento dei nuovi valori calcolati mediante
    interpolazione
412         A_XPos = A_XEnd.update();
413         A_YPos = A_YEnd.update();
414         A_ZPos = A_ZEnd.update();
415         A_XPos_turn=-A_XPos; //inverto i gradi per permettere di
    ottenere il verso opposto
416
417         B_XPos = B_XEnd.update();
418         B_YPos = B_YEnd.update();
419         B_ZPos = B_ZEnd.update();
420         B_XPos_turn=-B_XPos;
421         if(turnRight | turnLeft){ //per girare e' necessario che
    3 gambe si muovano in verso opposto alle rimanenti
422           L1.newAngle(A_XPos_turn,A_YPos,A_ZPos,&L1theta1,&
    L1theta2,&L1theta3);
423           L2.newAngle(B_XPos_turn,B_YPos,B_ZPos,&L2theta1,&
    L2theta2,&L2theta3);
424           L3.newAngle(A_XPos_turn,A_YPos,A_ZPos,&L3theta1,&
    L3theta2,&L3theta3);
425           L4.newAngle(B_XPos,B_YPos,B_ZPos,&L4theta1,&L4theta2,&
    L4theta3);
426           L5.newAngle(A_XPos,A_YPos,A_ZPos,&L5theta1,&L5theta2,&
    L5theta3);
```

```
427         L6.newAngle(B_XPos,B_YPos,B_ZPos,&L6theta1,&L6theta2,&
    L6theta3);
428       }
429       else{
430         L1.newAngle(A_XPos,A_YPos,A_ZPos,&L1theta1,&L1theta2,&
    L1theta3);
431         L2.newAngle(B_XPos,B_YPos,B_ZPos,&L2theta1,&L2theta2,&
    L2theta3);
432         L3.newAngle(A_XPos,A_YPos,A_ZPos,&L3theta1,&L3theta2,&
    L3theta3);
433         L4.newAngle(B_XPos,B_YPos,B_ZPos,&L4theta1,&L4theta2,&
    L4theta3);
434         L5.newAngle(A_XPos,A_YPos,A_ZPos,&L5theta1,&L5theta2,&
    L5theta3);
435         L6.newAngle(B_XPos,B_YPos,B_ZPos,&L6theta1,&L6theta2,&
    L6theta3);
436       }
437
438
439       // alla fine del calcolo di un singolo step di
    interpolazione viene eseguito il successivo
440       if ((Go==false | A_XEnd.isFinished()==true)){
441         if(forward | turnLeft){ //per eseguire il movimento in
    avanti e girare a sinistra si incrementano gli step che
    potrano scorrere la matrice definita trajectory
442           commandStepA++;
443           commandStepB++;
444           if (commandStepA> 11){
445             commandStepA = 0;
446           }
447           if (commandStepB > 11){
448             commandStepB = 0;
449           }
450         }
451         if(backward | turnRight){//per eseguire il movimento
    all'indietro e girare a destra si decrementano gli step che
    potrano scorrere la matrice definita trajectory
452           commandStepA--;
453           commandStepB--;
454           if (commandStepA<=0){
455             commandStepA = 11;
456           }
457           if (commandStepB<=0){
458             commandStepB = 11;
```

```
459             }
460           }
461           xMov = trajectory[commandStepA][0];
462           yMov=0; //per la traiettoria ci si sposta lungo il
    piano XZ mentre il valore di y rimane nullo,rimane percio' il
    valore di offset fissato
463           zMov = trajectory[commandStepA][1];
464           int duration = trajectorySpeed;
465           A_XEnd.go(xMov,duration);    //funzione che calcola l'
    interpolazione per il nuovo valore con il tempo per il
    movimento
466           A_YEnd.go(yMov,duration);
467           A_ZEnd.go(zMov,duration);
468           xMov = trajectory[commandStepB][0];
469           zMov = trajectory[commandStepB][1];
470           duration = trajectorySpeed;
471           B_XEnd.go(xMov,duration);
472           B_YEnd.go(yMov,duration);
473           B_ZEnd.go(zMov,duration);
474           Go =true;
475         }
476       }
477     }
```

Listing A.1: Intero codice Arduino

Appendice B

Layout del circuito e foto hexapod

B.1 Layout del circuito

Viene riportato il circuito PCB realizzato mediante software EasyEDA.Il PCB è stato progettato come shield da poter essere facilmente inserita sopra la scheda Arduino risparmiando spazio.

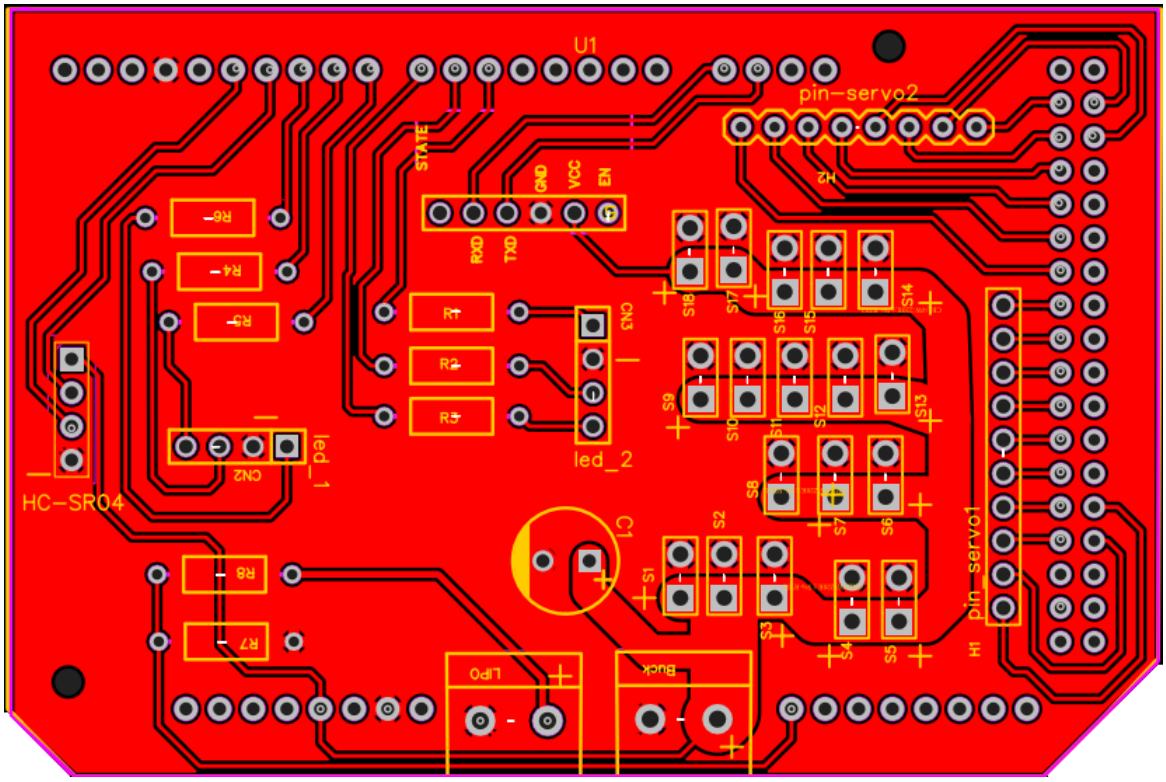

Figura B.1: PCB Layout

B.2 Foto hexapod

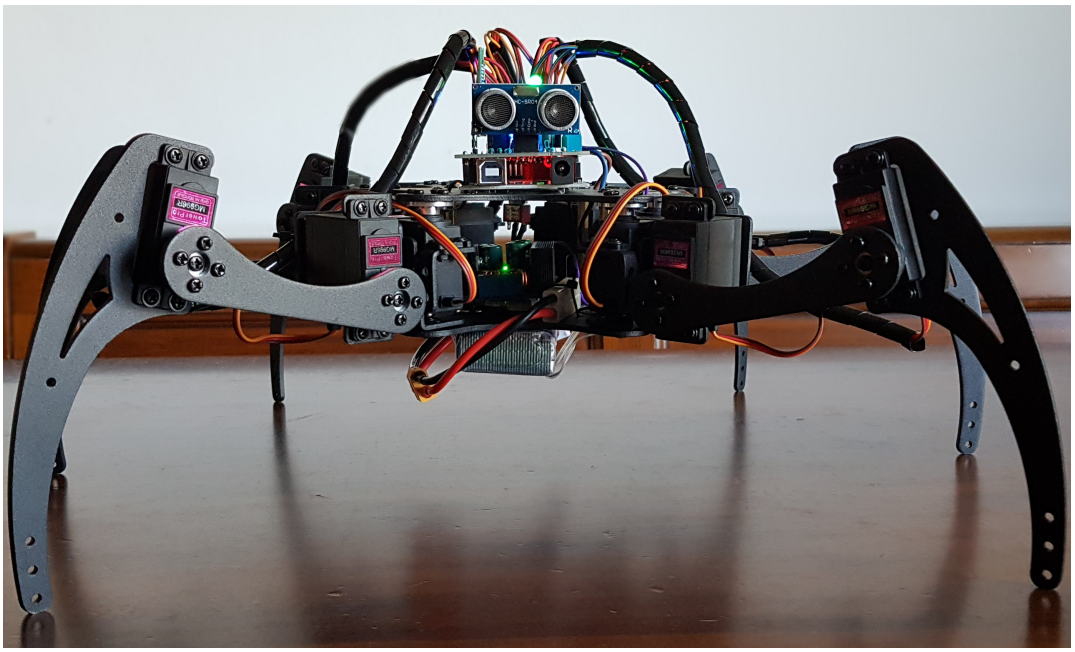

Figura B.2: Foto frontale

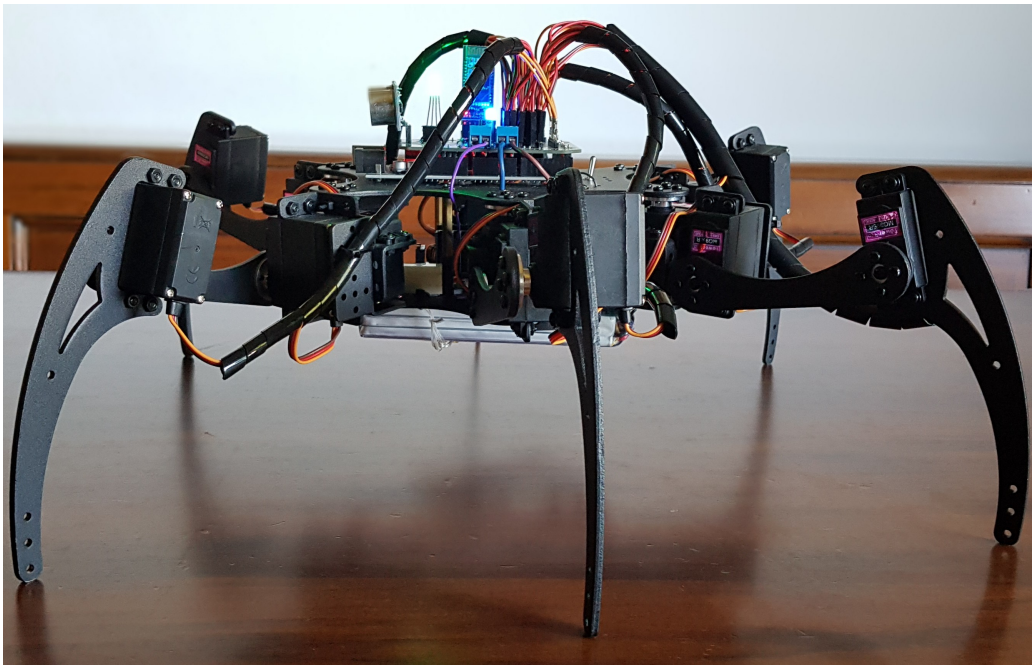

Figura B.3: Foto laterale

Bibliografia

[1] Tenreiro Machado, José & Silva, Manuel,(2006),An Overview of Legged Robots.

[2] MG996R datasheet,https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf.

[3] I servomotori,28 Aprile 2020 da ne555, https://www.ne555.it/i-servomotori/.

[4] Alimentatori switching,07/03/2022, https://leonardocanducci.org/wiki/tp4/alimentatori_switching.

[5] Modulo convertitore seriale/Bluetooth HC-05,https://www.robotstore.it/en/Modulo-convertitore-Seriale-Bluetooth-HC-05.

[6] Ultrasonic Ranging Module HC-SR04 datasheet,https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

[7] Arduino Mega 2560 datasheet,https://docs.arduino.cc/static/3f797d7ef22182a4e973bb7aafd3148d/A000067-datasheet.pdf.

[8] Hexi robot,Common Walking Gaits For Hexapods,Posted on November 20,2015 by Mithi,https://hexyrobot.wordpress.com/2015/11/20/common-walking-gaits-for-hexapods/.

[9] Ma J, Qiu G, Guo W, Li P, Ma G. Design, Analysis and Experiments of Hexapod Robot with Six-Link Legs for High Dynamic Locomotion. Micromachines. 2022; 13(9):1404.https://doi.org/10.3390/mi13091404.

[10] Y. Zhai et al., "Gait planning for a multi-motion mode wheel-legged hexapod robot," 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO), Qingdao, China, 2016, pp. 449-454, doi: 10.1109/ROBIO.2016.7866363.

[11] Marco Gabiccini,Cinematica diretta ed inversa di manipolatori seriali,Robotica 1,A.A.2009/2010, http://docenti.ing.unipi.it/gabiccini-m/RAR/04%20-%20Cinematica%20diretta%20ed%20inversa%20manipolatori%20seriali.pdf.

[12] K. Priandana, A. Buono and Wulandari, "Hexapod leg coordination using simple geometrical tripod-gait and inverse kinematics approach," 2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Bali, Indonesia, 2017, pp. 35-40, doi: 10.1109/ICACSIS.2017.8355009.

Ringraziamenti

In conclusione a questa tesi desidero ringraziare in primo luogo la mia famiglia che mi ha sempre sostenuto e dato fiducia lungo questo percorso accademico.Voglio esprimere la mia profonda riconoscenza ai miei genitori e a mio fratello,che hanno sempre creduto nelle mie capacità e mi hanno spinto a perseguire i miei sogni.
In ultima vorrei ringraziare di cuore i miei amici,con i quali ho condiviso le aule e le lunghe ore di studio.