POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

LIONHELL MCMILLAN II: RIPROGETTAZIONE DI UN ROBOT ESAPODE BIOLOGICAMENTE ISPIRATO PER AREE MORFOLOGICAMENTE INSTABILI

AI & R Lab

Laboratorio di Robotica e Intelligenza Artificiale

del Politecnico di Milano

Relatore: Prof. Giuseppina GINI
Correlatore: Ing. Vittorio LUMARE

Tesi di Laurea di:
Alessandro ROSINA matr. 783146

Anno Accademico 2014-15

Sommario

I robot stanno diventando parte integrante della vita dell'uomo, svolgono un ruolo fondamentale nei processi di automazione industriale, la tecnologia pervade le nostre case e i primi robot esplorano lo spazio e si muovono su terreni sconosciuti.

In particolare, lo sviluppo di robot in grado di esplorare l'ambiente circostante, di muoversi all'interno di ruderi pericolanti o in stretti cunicoli sta diventando una realtà sempre più imprescindibile, e di conseguenza si è resa necessaria la creazione di nuovi sistemi di locomozione e di design alternativi per superare più facilmente ostacoli di grandi o piccole dimensioni.

Lo scopo della tesi è quello di potenziare e migliorare il robot LionHell, rendendolo capace di esplorare con maggiore efficienza il terreno intorno a lui e migliorandone conseguentemente l'usabilità in ambienti aperti o chiusi, su terreni accidentati o non.

LionHell II è un robot esapode dotato di un sistema di locomozione chiamato Wheg, costituito da un asse centrale rotativo e da tre gambe unite nel giunto centrale ed equidistanti: I nuovi Wheg e il giunto passivo centrale che è stato aggiunto garantiscono un movimento più fluido quando il robot si appresta a curvare. La barra sensoriale e la coda sono state a loro volta rinforzate, e la forza sviluppata dalla coda è stata incrementata in modo da affrontare più facilmente un ostacolo in cui sia richiesto il suo intervento. Il controllo remoto per mezzo di un telecomando wireless garantisce un controllo totale sul movimento del robot stesso e l'aggiunta finale di alcuni aspetti estetici rendono il robot più facilmente accettabile da parte di un pubblico esterno, svolgendo in parte il ruolo di protezioni in caso di ribaltamento accidentale.

Abstract

Robots are becoming an integral part of human life, playing a key role in the processes of industrial automation, technology pervades our homes and the first robots explore the space and move in unknown territory.

In particular, the development of robots that can explore their surroundings, that can move within crumbling ruins or in narrow tunnels is becoming an increasingly essential, and therefore it was necessary to create new systems of locomotion and alternative design to more easily overcome obstacles of large or small size.

The aim of the thesis is to enhance and improve the robot LionHell, enabling it to more efficiently explore the terrain around him and consequently improving the usability in open or closed, on rough terrain or not.

LionHell II is a hexapod robot equipped with a locomotion system called Wheg, consisting of a central rotary axis and three legs joined in the central joint and equidistant: the new Wheg and the passive central joint that has been added guarantee a smooth motion when the robot is about to bend. The sensory bar and tail were in turn reinforced, and the force developed by the tail was increased in order to deal more easily an obstacle where it is required his intervention. The remote control by means of a wireless remote control ensures total control over the movement of the robot itself and the final addition of some aesthetic aspects make the robot more easily acceptable by an external public, performing in part the role of protections in case of accidental tipping.

Ringraziamenti

Desidero ringraziare la professoressa Giuseppina Gini e l'Ing. Vittoro Lumare, in qualità rispettivamente di Relatore e Correlatore, per avermi supportato ed incoraggiato nel progetto e nella stesura della tesi, senza il loro aiuto questa tesi non esisterebbe.

Un grande, enorme ringraziamento a Vittorio, che mi ha saputo dare dei consigli fondamentali e mi ha aiutato durante tutte le fasi del progetto di tesi, è sempre stato disponibile e mi ha sempre incoraggiato e sostenuto.

Sono grato al professore Andrea Bonarini e a tutte le persone del laboratorio AI & R Lab di Lambrate che mi hanno consigliato e che lavorando ciascuno al loro progetto hanno fatto di questo laboratorio un posto interessante e stimolante.

Vorrei dedicare questo progetto di tesi alle persone a me più care, alle mie nonne Mimì e Nella, che non hanno avuto la possibilità di vedere il completamento del corso di laurea, ai miei genitori, che mi hanno sempre aiutato e supportato e per essermi stati vicino ogni momento durante tutti questi anni di lavoro e alla mia ragazza Julia, che si è letta tutta la tesi e mi ha sempre sostenuto anche durante i miei momenti più difficili.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

I robot stanno diventando una parte integrante della società e della cultura dell'era moderna, e vivere senza e rifiutare la nuova e crescente tecnologia riduce considerevolmente le aspettative di vita e la qualità della vita stessa.

I robot nati come robot di esplorazione, quali il robot aspirapolvere Roomba, piuttosto che l'ultimo Rover della NASA Opportunity, hanno in comune una cosa: sono stati progettati per poter affrontare e analizzare l'ambiente in cui si sarebbero trovati a lavorare e, nel caso, affrontare o eludere gli ostacoli incontrati lungo il loro percorso.

Lo scopo della tesi, in questo caso, è quello di potenziare e migliorare un robot preesistente, LionHell, progetto di tesi di Vittorio Lumare [50, 49], anno accademico 2011-2012, in modo da renderlo più facilmente controllabile tramite l'utilizzo di un telecomando a controllo remoto, migliorandone il sistema di locomozione e la coda, facilitargli il movimento per mezzo di un giunto centrale passivo e renderlo accettabile esteticamente.

L'obiettivo è quello di creare un robot che sia in grado, un giorno, di esplorare zone con terreni disconnessi, quali ad esempio tunnel o edifici diroccati e, con le dovute modifiche, possa persino contribuire nell'esplorazione di altri pianeti, sfruttando il sistema di navigazione basato sulla combinazione di Wheg e coda per poter scavalcare facilmente ostacoli di grandi dimensioni.

I robot esplorativi stanno ricoprendo un ruolo sempre più importante, a partire dai robot aspirapolvere Roomba [15, 3, 53] fino ai più avanzati rover della NASA Spirit e Opportunity [8]. La necessità di esplorare terreni sconosciuti e accidentati ha portato alla ricerca di nuovi sistemi di locomozione, e tra questi i robot con Wheg (Prolero [52], Climbing Mini Whegs [2, 68, 29, 51], EMBOT [38], Lunar Whegs [33, 5, 34], Mini-Whegs IV [55], OutRunner [26], Ratasjalg [7, 60], Rhex [16, 59, 39], Termes [67, 66, 57], USAR Whegs [43, 44, 9]) hanno iniziato a diffondersi sempre più rapidamente, grazie al fatto che un Wheg, costituito da un solo attuatore che agisce su un asse centrale rotativo a cui sono connesse una o più barre che svolgono la funzione di gambe, risulta più semplice da controllare e più economico rispetto ad una gamba, che deve necessariamente essere costituita da almeno tre attuatori che ne controllano i giunti motorizzati. Dalla nascita del primo robot dotato di Wheg, Prolero [52], si sono susseguite nuove ricerche nell'ambito di robot equipaggiati con questo sistema di locomozione, come ad esempio Rhex [16, 59, 39], basato sull'utilizzo di sei Wheg ad una gamba che si muovono alternativamente per superare gli ostacoli che gli si presentano contro, o Termes [67, 66, 57] che sfrutta quattro piccoli Wheg a tre gambe per scalare le sue stesse costruzioni.

Il robot finale che è stato ottenuto al termine del progetto di questa tesi è stato chiamato LionHell Mc Millan II: successivamente si farà riferimento al nuovo robot solo con il nome di LionHell II, mentre con il nome di LionHell ci si riferisce al precedente progetto di Vittorio Lumare.

LionHell II è un robot esapode dotato di Wheg a tre gambe ispirato in parte al mondo animale. Il robot è dotato di una barra sensoriale con 4 telemetri a infrarossi, di cui tre rivolti verso il terreno in modo da analizzare la conformazione della superficie che LionHell II si appresta ad affrontare, mentre uno rivolto di fronte a sé, per individuare la presenza di ostacoli e controllare se l'eventuale ostacolo è affrontabile oppure no. La barra sensoriale della testa e il comportamento dei sensori è ispirato alle vibrisse dei felini [48, 50], ossia i baffi dotati di terminazioni nervose, il cui ruolo è quello di analizzare il terreno e la presenza di ostacoli, particolarmente utili di notte. Il sistema di locomozione è costituito dall'utilizzo di Wheg, ossia di un asse rotativo

centrale a cui sono connesse, in questo caso, tre gambe equidistanti tra loro, in modo da creare un angolo di 120° tra una gamba e l'altra. I Wheg permettono a LionHell II di superare con facilità ostacoli di piccole o grandi dimensioni [65, 11], e l'aggiunta di una coda evita che LionHell II si possa ribaltare quando è costretto, ad esempio, a superare una serie di scalini, o quando in generale deve affrontare ostacoli di grandi dimensioni in cui solo i Wheg frontali e quelli posteriori sono in grado di fare forza. La coda che LionHell II possiede prende spunto dal geco Cosymbotus Platyurus [17]: la coda infatti non si limita ad immagazzinare il grasso e a garantire l'equilibrio e un appiglio, ma agisce anche come gamba di emergenza durante le scalate o nel caso di superfici molto scivolose. In origine LionHell non era in grado di curvare, in quanto il sistema sensoriale non era sufficiente per analizzare l'ambiente circostante e prendere le decisioni del caso.

Le modifiche apportate hanno permesso di migliorare il sistema di locomozione, chiamato Wheg, che è stato modificato in modo da adattarsi più facilmente al terreno e in modo da ridurre contraccolpi dovuti al suolo accidentato. I nuovi Wheg che sono stati aggiunti sono dotati di uno strato di gommapiuma che riduce i sobbalzi e di uno strato in gomma a diretto contatto con il terreno, che garantisce una buona aderenza su qualunque superficie.

L'utilizzo di un giunto centrale passivo permette una maggiore maneggiabilità durante le curve e garantisce un migliore adattamento del robot su terreni accidentati [47], accompagnando il moto dei Wheg e facilitandone dunque il movimento su qualunque superficie.

la coda è stata rinforzata e la potenza dei motori è stata incrementata in modo da favorire il robot durante le scalate più difficili, come se fosse una gamba di emergenza, impedendogli di rovesciarsi e mantenendolo saldamente a contatto con il terreno. La nuova coda garantisce una maggiore stabilità in salita e un'elevata resistenza agli urti.

Il controllo remoto è stato ottenuto per mezzo dell'utilizzo di due XBee [37, 20], uno installato su LionHell II e l'altro sul telecomando, che comunicano via wireless e permettono una buona maneggevolezza anche grazie al telecomando che per mezzo di accelerometri è in grado di trasmettere la dire-

zione desiderata semplicemente inclinandolo. Il controllo remoto è stata una modifica fondamentale per LionHell II, in quanto ha dato la possibilità di discriminare la direzione del robot, operazione prima impossibile utilizzando solamente la barra sensoriale preesistente, che risulta insufficiente per esplorare l'ambiente circostante ed essere in grado di effettuare scelte consapevoli sulla strada da percorrere.

LionHell II ha subito anche alcune modifiche in modo da modificarne l'aspetto esteriore, rendendolo più facilmente accettabile da un pubblico esterno [64, 54], aggiungendo un volto animale ed una corazza, che svolge sia un ruolo estetico sia il ruolo di proteggere il busto, la scheda di controllo e l'XBee, garantendo una sicurezza extra nel caso sfortunato in cui il robot dovesse capovolgersi.

In conclusione LionHell II si è dimostrato capace di superare ostacoli di varie dimensioni, dai sei ai sedici centimetri, è in grado di superare con facilità delle rampe, anche affrontandole dalla direzione sbagliata, è in grado di curvare facilmente intorno al proprio asse sia in senso orario sia in senso antiorario e di superare un terreno disseminato di oggetti sparsi casualmente sul pavimento.

Equipaggiato con una telecamera e delle torce, LionHell II sarebbe in grado di esplorare edifici pericolanti o condotti molto stretti e in generale potrebbe essere utilizzato in ogni ambito in cui è richiesto un robot con discrete capacità di scalare. Studi recenti [61, 25] hanno inoltre dimostrato la possibilità di utilizzare robot esplorativi per applicazioni swarm, ossia per la creazione di numerosi robot economici capaci di collaborare tra loro per la creazione di una mappa comune della zona o per effettuare operazioni complesse.

La tesi è strutturata nel modo seguente:

Nel Capitolo 2 si descrive lo stato dell'arte fino ad ora, introducendo il significato di Wheg e mostrando esempi di robot che ne sono dotati, per terminare con la descrizione del progetto di tesi di Vittorio Lumare, che descrive lo stato del robot originale, mostrando le caratteristiche generali e motivando le ragioni che ci hanno portato ad effettuare le modifiche.

Nel Capitolo 3 si descrivono brevemente le modifiche che sono state effettuate a LionHell II e si mostrano i componenti che hanno svolto un ruolo chiave nella scelta delle operazioni da effettuare.

Nel Capitolo 4 si descrivono in dettaglio le modifiche meccaniche effettuate, i Wheg che sono stati sostituiti, il ruolo del giunto centrale passivo e l'impostazione della coda e la ragione della sua presenza.

Nel Capitolo 5 si descrive il sistema del controllo remoto, il funzionamento dell'XBee e le modifiche che sono state apportate al firmware affinché il telecomando e LionHell II fossero in grado di comunicare.

Nel Capitolo 6 si mostra l'aspetto esteriore di LionHell II, descrivendone il ruolo e l'aspettativa che gli umani hanno nei confronti di un robot con un aspetto meccanico confronto ad un robot con aspetto umanoide od animale, e descrivendo il volto di LionHell II, la corazza e in generale le modifiche che sono state apportate per renderlo più gradevole.

Nelle Conclusioni 7 si riassumono i risultati, le valutazioni di questi e le prospettive future, mostrando alcuni possibili sviluppi futuri di LionHell II e le modifiche che possono essere ulteriormente effettuate.

Nell'Appendice A si mostra il firmware della scheda di controllo CM-510.

Nell'Appendice B vi è il manuale utente, che descrive i passi necessari per riprogrammare LionHell II, i cavi e la componentistica necessaria e tutte le informazioni del caso, indicando anche una serie di indirizzi URL utili.

Nell'Appendice C si mostrano i datasheet di tutti i componenti di LionHell II e del telecomando, i cavi di connessione, di programmazione e dell'XBee.

Capitolo 2

Stato dell'arte

I robot hanno da sempre affascinato l'uomo, e in particolare la creazione di oggetti inanimati che, nelle leggende, avvivati da complessi meccanismi, dalla volontà di suscettibili divinità o per mezzo di potenti stregonerie, erano in grado di eseguire i comandi dei loro creatori.

Alcune di queste leggende sono pervenute fino a noi, come ad esempio la leggenda inuit del Tupilak [36, 35] (dall'inuktitut ᑐᐱᓪᓚᒃ [12]), nato come creatura inanimata a partire da parti di animali, animata dalla stregoneria e messa in mare in modo che potesse cercare ed eliminare la vittima designata (ma attenzione: uno stregone abbastanza potente sarebbe stato in grado di riprogrammare la creatura facendo sì che tornasse indietro ad uccidere il suo stesso creatore).

Allo stesso modo, la leggenda ebraica del Golem [45] (dall'ebraico גולם [14]) parla della creazione di una creatura a partire da un corpo in argilla e animata tramite la parola "verità" (dall'ebraico אמת [14]), dotata di straordinaria forza e resistenza ma priva della parola, del pensiero e di qualunque capacità di provare emozioni, in quanto priva dell'anima. In particolare, è famosa la vicenda del Golem di Praga [45], evocato dal rabbino Jehuda Löw ben Bezalel, in cui il rabbino perde il controllo della creatura, la quale inizia a distruggere tutto sul suo cammino, fino a che non viene fermata definitivamente dal suo stesso creatore per mezzo della parola "morto" (in ebraico מת [14]).

Figura 2.1: Il Tupilak e il Golem

(a) Tupilak (b) Golem

Successivamente, si iniziarono a costruire i primi modelli di robot, a partire da Leonardo da Vinci [63], il quale progettò, intorno al 1495, un cavaliere meccanico capace di alzarsi in piedi, agitare le braccia e muovere la testa e la mascella, ma sarà solo a partire dal 1738 che verrà costruito il primo robot funzionante, un androide in grado di suonare un flauto, fabbricato da Jacques de Vaucanson [10].

Con il passare dei secoli, la creazione di robot in grado di esplorare il mondo circostante, scalare degli ostacoli o effettuare anche movimenti più complessi come scavare, raccogliere dei campioni o analizzare delle rocce diventa sempre più realistica e le funzionalità dei robot aumentano di conseguenza.

Il robot Roomba, ad esempio, nasce come robot pulitore ed aspirapolvere nel 2002 [3], capace di analizzare la stanza, la presenza di ostacoli da evitare o scale da cui potrebbe cadere, e il crescente interesse da parte dei programmatori nei confronti del Roomba ha portato allo sviluppo di iRobot Create [15, 53, 3], dove i motori di aspirazione sono sostituiti da un vano su cui è possibile installare telecamere, laser o altri componenti.

Nel 2004 i due robot gemelli Spirit e Opportunity [8] sono giunti sul pianeta rosso Marte, ed hanno iniziato a raccogliere dati tecnici e ad inviare immagini, esplorando il terreno marziano e percorrendo centinaia di chilometri,

passando da un cratere ad un altro ed inviando tutte le informazioni ricevute sulla Terra.

L'esplorazione da parte di robot mobili è ancora agli inizi, e le loro funzionalità e capacità continuano ad aumentare e migliorare con il corso degli anni: la necessità di trovare metodi di navigazione alternativi è fondamentale affinché un giorno sia possibile creare robot in grado di muoversi autonomamente e di affrontare gli ostacoli che gli si presentino.

La Sezione 2.1 descrive il sistema di locomozione che è stato usato per LionHell II, chiamato Wheg, definendone il significato e mostrandone la differenza rispetto a una ruota.

La Sezione 2.2 descrive i principali robot che utilizzano il sistema di locomozione Wheg, mostrando in particolare i Wheg su Prolero, Rhex e Termes.

La Sezione 2.3 mostra il robot LionHell prima che venisse modificato, al termine del progetto di tesi di Vittorio Lumare.

2.1 Wheg

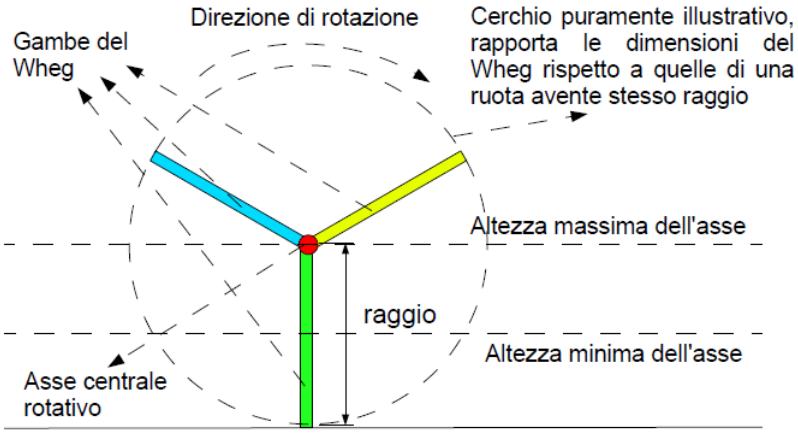
Il sistema di locomozione è l'elemento fondamentale che permette al robot di interfacciarsi ed esplorare l'ambiente circostante e richiede una scelta oculata che soddisfi i requisiti di maneggevolezza, semplicità meccanica e fluidità di movimento, garantendone, in questo caso, la possibilità di muoversi agevolmente su terreni accidentati, superando anche ostacoli di piccole o medie dimensioni.

La Sottosezione 2.1.1 definisce il significato di Wheg e ne descrive il funzionamento basilare.

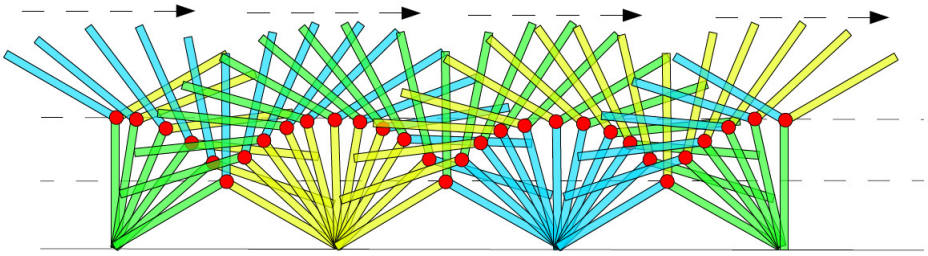
La Sottosezione 2.1.2 dimostra la superiorità dei Wheg contro una ruota avente stesso raggio, simulandone il comportamento contro vari ostacoli di altezza differente, e mostra i vantaggi e gli svantaggi derivanti dall'utilizzo dei Wheg.

Figura 2.2: Wheg a 3 gambe

(a) Descrizione



(b) In movimento

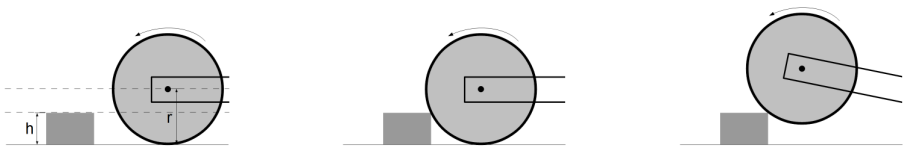


2.1.1 Definizione

I Wheg sono un meccanismo di locomozione per robot che combinano la semplicità di movimento di una ruota con le capacità di scalare e di superare ostacoli derivanti dall'utilizzo delle gambe [11, 58]. L'acronimo deriva infatti dalla combinazione delle parole wheel e leg (ruota e gamba) e meccanicamente consistono in un asse centrale rotativo a cui sono collegate una o più barre che svolgono la funzione di gambe.

Il funzionamento di un Wheg è estremamente semplice: considerando l'esempio in Figura 2.2a, le tre gambe sono connesse ad un asse centrale che ruota su se stesso e il punto di contatto con il terreno è costituito dall'estremità della gamba. Il Wheg in rotazione è mostrato in Figura 2.2b.

Figura 2.3: Ruota che affronta un ostacolo di altezza $h < r$



2.1.2 Wheg contro Ruota

La superiorità del Wheg confronto all'utilizzo di una semplice ruota è facilmente dimostrabile [65, 11]:

1. Si consideri una ruota che si appresti ad affrontare un ostacolo come in Figura 2.3 [65], posto sul pavimento, di molto più piccolo del raggio della ruota stessa;

2. Una volta che la ruota sarà entrata in contatto con l'ostacolo, la frizione che si genererà forzerà il punto di contatto a rimanere nella stessa posizione e mentre la torsione della ruota continuerà ad agire, il punto di contatto funzionerà come un pivot. Se la forza del motore risulterà sufficiente il risultato sarà che la ruota supererà l'ostacolo proseguendo nel suo percorso;

3. Si consideri ora una situazione analoga [65], in cui però l'ostacolo che la ruota va affrontando ha la stessa altezza del suo raggio ($h \approx r$) come in Figura 2.4. In questo caso il punto di contatto è sul lato dell'ostacolo, e non sopra come nel caso precedente e la frizione che si dovrebbe venire a creare affinché possa essere scalato dovrebbe essere tale da permettere al robot di muoversi in maniera verticale rispetto all'ostacolo. Tipicamente questa situazione non è uno scenario realistico e il risultato dell'esperimento sarà che la ruota inizierà a slittare sul posto, cambiando continuamente il punto di contatto con l'ostacolo. Dato che la ruota fallirebbe in questo scenario, la ruota fallirebbe per tutti gli scenari in cui $h > r$ in quanto il punto di contatto risulterà sempre sul lato dell'ostacolo e non sopra.

Figura 2.4: Ruota che affronta un ostacolo di altezza $h \approx r$

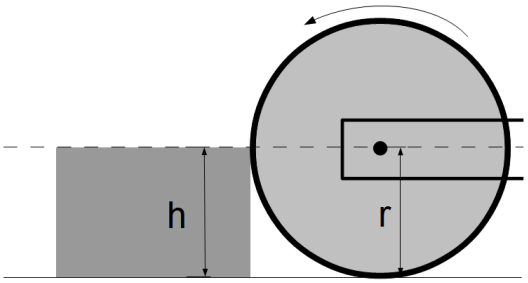
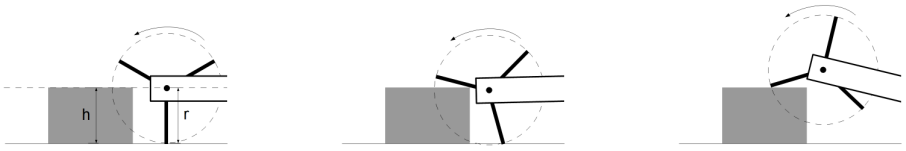


Figura 2.5: Wheg che affronta un ostacolo di altezza $h \approx r$



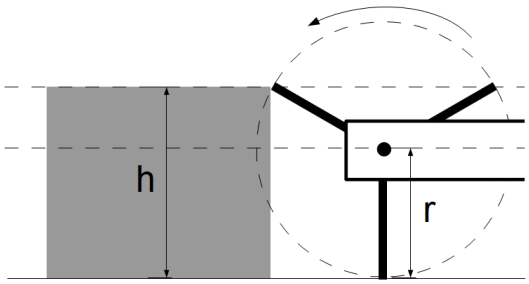
Si riconsideri ora l'esempio precedente [65], in cui $h \approx r$, ma dove al posto della ruota sia presente un Wheg a tre gambe (nella Figura 2.5 si può osservare la presenza di un cerchio attorno al Wheg: tale cerchio è puramente illustrativo e serve esclusivamente a rapportare le dimensioni del Wheg rispetto a quelle della ruota):

1. Per prima cosa si può osservare l'enorme spazio vuoto che la struttura presenta, spazio che verrà utilizzato a proprio vantaggio: la struttura è infatti in grado di sfruttare lo spazio vuoto, affrontando l'ostacolo dall'alto e non dal lato come nel caso della ruota;

2. La forza esercitata dai motori e la torsione del Wheg faranno il resto, permettendo di sfruttare il punto di contatto come pivot per alzare il

telaio del Wheg come mostrato nella Figura 2.5 e superare l'ostacolo (si ricorda sempre che il cerchio serve solamente per mostrare la traiettoria delle tre gambe);

Figura 2.6: Wheg che affronta un ostacolo di altezza $h \approx \frac{3}{2}r$



3. Si riconsideri ora un nuovo esempio [65], mostrato in Figura 2.6 , in cui abbiamo $h > r$, ed in particolare il caso in cui $h \approx \frac{3}{2}r$ (in quanto sono presenti 3 gambe). In questo caso estremo il Wheg non sarà in grado di superare l'ostacolo: la capacità di scalarlo dipende da quanto il Wheg è capace di penetrare il profilo dell'ostacolo. È possibile ottenere tale obiettivo riducendo l'angolo tra le due gambe superiori, ma questo comprometterebbe la struttura del Wheg rendendolo instabile.

Da questa analisi è possibile evidenziare i principali vantaggi e svantaggi derivanti dall'utilizzo di un Wheg:

Pro la capacità di scalare ostacoli di altezza maggiore rispetto a quelli affrontati da una ruota avente medesimo raggio;

Pro la velocità di movimento del robot si mantiene elevata;

Pro la maggiore semplicità di costruzione e di controllo rispetto ad una gamba, che deve necessariamente essere controllata da due o tre attuatori;

Contro Il terreno su cui si muove deve essere ruvido, in modo da poter fare forza ed essere in grado di superare l'ostacolo;

Contro Delle gambe troppo sottili potrebbero sprofondare su superfici morbide o non rigide, quali la sabbia o il fango, a causa della superficie di contatto ridotta;

Contro Se le gambe colpissero parti mobili, come lunghi fili d'erba o cavi molto sottili, il Wheg potrebbe rimanerne attorcigliato, bloccandosi e rischiando di danneggiare il robot nel caso in cui i motori cercassero di fare troppa forza per liberarsi.

2.2 Robot dotati di Wheg

Il Wheg è un sistema di locomozione utilizzato in moltissimi robot esplorativi [11], a partire da Prolero [52], il primo robot in assoluto ad essere equipaggiato con Wheg, mostrato in Figura 2.7a, è un robot dotato inizialmente di 4 Wheg e successivamente di 6 Wheg ad 1 gamba, è stato sviluppato da Martin-Alvarez A., de Peuter P., Hillebrand JR., Putz P., Matthyssen A. e de Weerd J. nel 1996, è descritto in dettaglio nella Sottosezione 2.2.1. Successivamente, moltissimi altri robot (Figura 2.7) hanno seguito il suo esempio:

- Climbing Mini Whegs [2, 68, 29, 51], mostrato in Figura 2.7b, è un robot dotato di 4 Wheg a 4 gambe capaci di aderire su varie superfici, è stato sviluppato presso Case Western Reserve University;
- Embot [38], mostrato in Figura 2.7c, è un robot dotato di 4 Wheg a 3 gambe, è stato sviluppato presso il Politecnico di Milano da Gaibotti A. e Mariggiò F. nel 2011;
- Lunar Whegs [33, 5, 34], mostrato in Figura 2.7d, è un robot dotato di 6 Wheg a 3 gambe, è stato sviluppato da Dunker P.A. nel 2009;

- Mini-Whegs IV [55], mostrato in Figura 2.7e, è un robot di piccole dimensioni dotato di 4 Wheg a 3 gambe, è stato sviluppato da Morrey J., Lambrecht B., Horchler A., Ritzmann R. e Quinn R. nel 2003;
- OutRunner [26], mostrato in Figura 2.7f, è un robot da corsa dotato di 2 Wheg a 3 gambe, è stato sviluppato da Cotton S., Black C., Payton N., Ford K., Howell W. e Conrad, J. nel 2014;
- Ratasjalg [7, 60], mostrato in Figura 2.7g, è un robot dotato di 2 soli Wheg a 6 gambe, che in caso di bisogno possono trasformarsi in ruote, è stato sviluppato e brevettato da Sell R. alla Tallin University of Technology in Estonia nel 2007;
- Rhex [16, 59, 39], mostrato in Figura 2.7h, è un robot con 6 Wheg ad 1 gamba, è stato sviluppato da Altendorfer R., Moore N., Komsuoglu H., Buehler M., Jr Brown H., McMordie D., Saranli U., Full R. e Koditschek D nel 2001, è descritto in dettaglio nella Sottosezione 2.2.2;
- Termes [67, 66, 57], mostrato in Figura 2.7i, è un robot con 4 Wheg a 3 gambe, è stato sviluppato da Werfel J., Petersen K., e Nagpal R. nel 2014 ed è descritto in dettaglio nella Sottosezione 2.2.3;
- USAR Whegs [43, 44, 9], mostrato in Figura 2.7j, è un robot dotato di 4 Wheg a 4 gambe, è stato sviluppato da Hunt A.J. presso Case Western Reserve University nel 2010.

Nelle sottosezioni successive descriveremo solo alcuni di questi robot, in particolare:

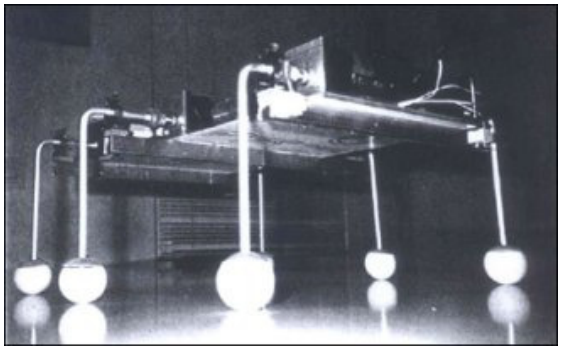
La Sottosezione 2.2.1 descrive il robot Prolero, il robot che per primo utilizzò la struttura dei Wheg.

La Sottosezione 2.2.2 descrive il robot Rhex, un tipo di robot simile sotto alcuni aspetti a Prolero, ma con maggiori capacità di movimento e di esplorazione.

La Sottosezione 2.2.3 introduce Termes, un robot di piccole dimensioni e discrete capacità di scalare.

Figura 2.7: Robot con Wheg

(a) Prolero

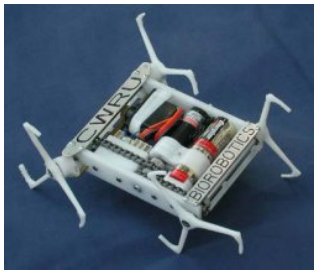


(b) Climbing Mini Whegs

(c) Embot

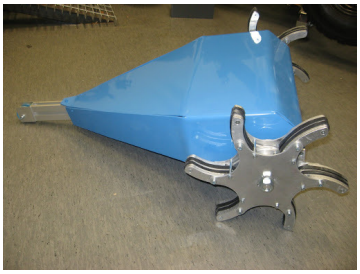
(d) Lunar Whegs

(e) Mini-Whegs IV



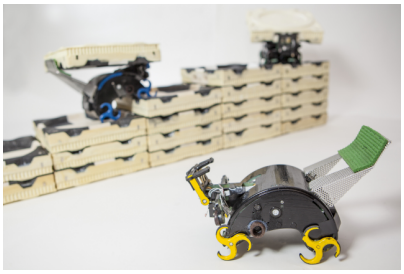
(f) OutRunner

(g) Ratasjalg

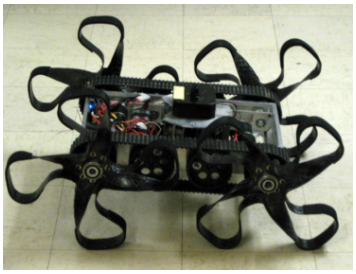


(h) Rhex

(i) Termes

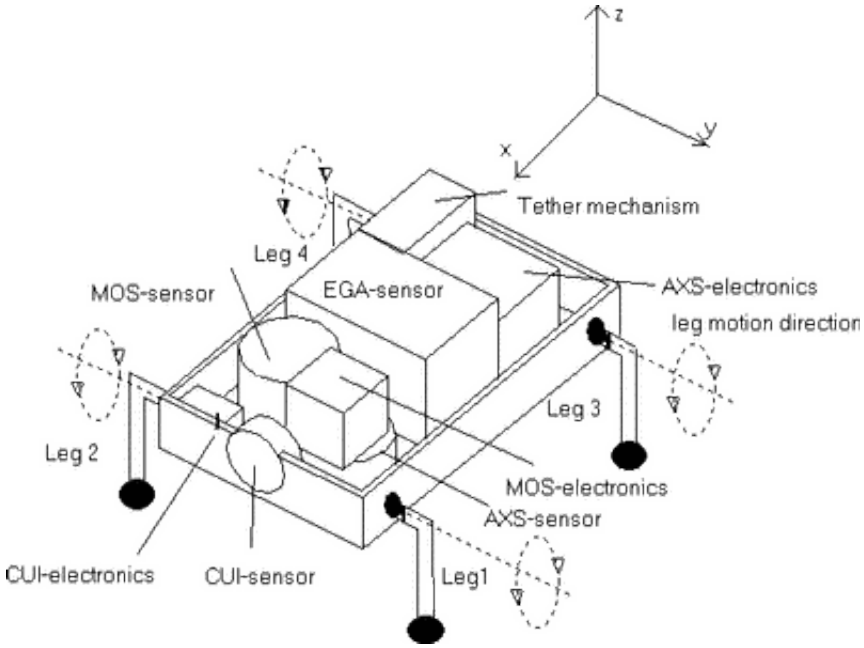


(j) USAR Whegs



2.2.1 Prolero

Figura 2.8: Prolero: modello schematico

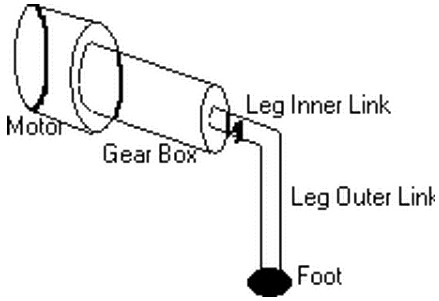


La prima versione di Wheg apparve nel robot Prolero [52](PROtotype of LEgged ROver, che significa prototipo di rover con gambe), progettato nel 1996 alla Case Western Reserve University come possibile robot esploratore di Marte.

Il robot, come mostrato nella Figura 2.8 è dotato di quattro gambe con forma a L, ciascuna delle quali dotata di un solo grado di libertà, il che permette di ridurre considerevolmente la complessità meccanica totale rispetto a quella di una gamba standard, che in genere richiede dai 2 ai 3 attuatori. Inoltre, la piccola dimensione di Prolero e la sua ottima capacità di carico in rapporto alle dimensioni ridotte (22x23x8 cm con una capacità di carico di 1.5 kg) ed in particolare la sua abilità nel superare senza problemi la maggior parte degli ostacoli che gli si presentano contro lo rendevano un ottimo candidato come possibile esploratore di Marte come mini robot mobile. Tale struttura fu sviluppata come conseguenza delle rigide norme imposte, norme che in

particolare andavano ad impattare sulla massa totale del robot, le dimensioni massime del robot a riposo e l'energia richiesta per il movimento.

Figura 2.9: Prolero: Wheg



Nella Figura 2.9 è possibile osservare in dettaglio la semplice struttura del Wheg, costituito da un singolo attuatore, un piccolo piede che consiste in una semplice sferetta di gomma e la gamba stessa ad L. La forma della gamba permette inoltre a Prolero di occupare meno spazio quando il robot è nella composizione compatta, adatta per il trasporto.

Il movimento dei Wheg di Prolero non è alternato, ma avviene simultaneamente: il risultato è che il robot si alza da terra con le quattro gambe dei Wheg per un attimo, si riabbassa e il corpo si ritrova a contatto con il terreno, le gambe dei Wheg ruotano di $\approx 180°$ e Prolero ricomincia a muoversi da capo.

2.2.2 Rhex

Oltre al famoso esempio di Prolero, sono presenti molti altri esempi di robot con Wheg, come nel caso di robot quali Rhex [16, 59, 58, 39] (Figura 2.10), equipaggiato con 6 Wheg (2 in più confronto a Prolero) ciascuno dei quali dotato di una sola gamba (come Prolero), ma stavolta ricurva (Figura 2.11), capace di muoversi alternativamente, e che gli garantisce la capacità di superare con facilità la maggior parte degli ostacoli.

A differenza di Prolero, Rhex non colpisce il terreno con il proprio corpo ogni volta che deve fare un passo, ma sfrutta il movimento alternato dei

Figura 2.10: Rhex

(a) Struttura esterna

(b) Struttura schematica

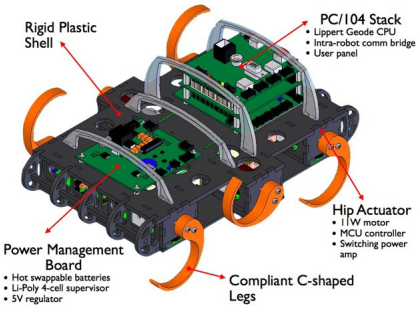
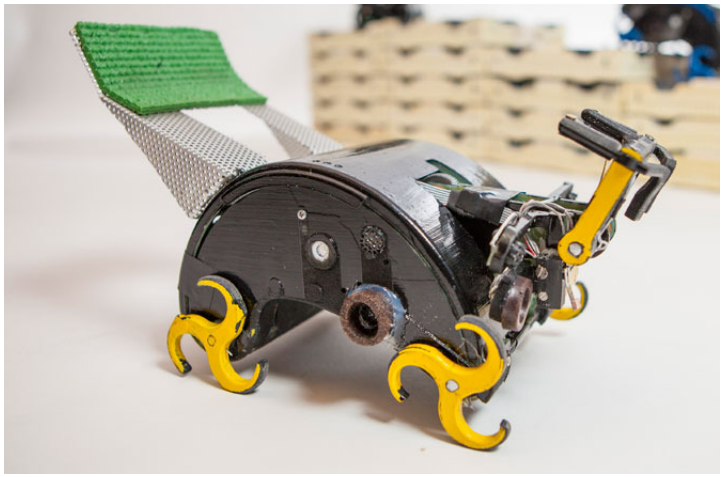


Figura 2.11: Rhex: Wheg

piedi dei Wheg [42], alterandone anche la velocità nel caso in cui il piede sia a contatto con il terreno oppure no. Il risultato è che la gamba, in posizione verticale e a contatto con il terreno, ruota intorno al perno centrale di $\approx 360°$, rallentando nuovamente quando entra nuovamente a contatto con il terreno, per poi ricominciare a muoversi.

2.2.3 Termes

Figura 2.12: Termes: struttura esterna



I robot Termes [67, 66, 57], ispirati alle termiti, sono robot progettati per collaborare tra loro e costruire piccoli edifici. Affinché i robot siano in grado di muoversi e scalare le loro costruzioni, sono stati provvisti di 4 Wheg, ciascuno delle quali dotato di tre minuscole gambe ricurve, garantendo loro discrete capacità di scalare.

A differenza di Rhex, la presenza di 3 gambe per ciascun Wheg garantisce un movimento più fluido ed un controllo maggiore durante il movimento, riduce i sobbalzi causati dall'impatto della gamba con il terreno e gli permette di scalare con facilità gli ostacoli.

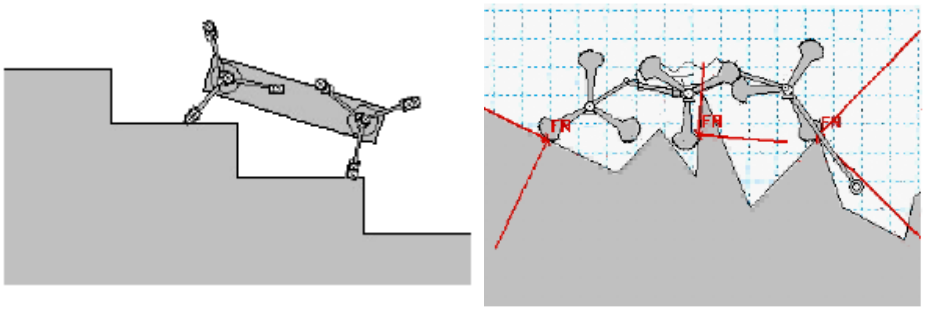
2.3 LionHell

LionHell è nato nel 2011 come progetto di tesi di Vittorio Lumare, per il Corso di Laurea Magistrale in Ingegneria Informatica [49, 50]. Questo lavoro è stato svolto presso AI & R Lab, il Laboratorio di Robotica e Intelligenza Artificiale del Politecnico di Milano.

Il primo progetto [49, 50], mostrato in Figura 2.13a, prendendo spunto dal robot Embot [38], prevedeva l'utilizzo di un corpo rigido e 4 Wheg con tre

Figura 2.13: LionHell: modelli iniziali

(a) Primo modello (b) Secondo modello



gambe l'uno, e l'obiettivo era quello di creare un robot che fosse in grado di navigare senza mappa e scavalcare e superare una serie di ostacoli di medie o grandi dimensioni (in Figura 2.13a Embot affronta, in simulazione, una serie di gradini). Il robot presentava però problemi a superare ostacoli di dimensioni maggiori, e tendeva a ribaltarsi, a causa della mancanza di un supporto retrostante.

Il secondo progetto [49, 50], mostrato in Figura 2.13b, ha introdotto un Wheg centrale, un nuovo collegamento aggiunto al corpo, un giunto centrale e una coda. La sezione del piede è stata inoltre modificata, in modo da avere una maggiore superficie di contatto ed esercitare una forza maggiore. Il nuovo robot così progettato ha dimostrato, in simulazione, di essere in grado di muoversi facilmente su terreni disconnessi e accidentati, ed è stato in grado di superare anche ostacoli di grandi dimensioni.

La prima reale implementazione di LionHell [49, 50], mostrata in Figura 2.14a, è costituita da un robot esapode provvisto di 6 Wheg, ciascuno dei quali dotato di 2 gambe disposte a 180° le une dalle altre, e da un sensore frontale. Il design del robot ha mostrato alcuni problemi: I Wheg con due gambe causavano la caduta del robot durante le scalate, inoltre erano troppo corte e il materiale di cui erano costituite era semplice plastica, che tendeva a rompersi facilmente.

Il progetto finale di LionHell[49, 50], mostrato in Figura 2.14c, ha aumentato la lunghezza totale delle gambe, alzando conseguentemente il centro di massa

del robot, ha allungato la coda in seguito alla nuova altezza e ha portato il numero di gambe per Wheg da 2 a 3. La struttura è stata rinforzata in parte con alcuni componenti in alluminio ed è stata aggiunta una barra sensoriale dotata di 4 telemetri che sostituisse il sensore visivo precedente, il quale si era dimostrato troppo lento ad analizzare il terreno di fronte a sé e quindi inefficace nel predire la presenza di ostacoli.

Il progetto finale LionHell Mc Millan di Vittorio Lumare prevede che il robot sia totalmente autonomo e che, una volta acceso, proceda dritto fino a che non incontra un ostacolo: a questo punto il sistema di controllo decide se affrontare l'ostacolo oppure cambiare direzione, e nel caso in cui il robot affronti l'ostacolo la scheda di controllo determina l'inclinazione della parte frontale del robot e la forza esercitata dalla coda.

LionHell è in grado di sollevare la parte anteriore del robot nel momento in cui si appresta ad affrontare un ostacolo: il sensore a infrarossi che punta in avanti svolge il ruolo di analizzare la presenza di ostacoli frontalmente al robot, e ne alza conseguentemente la parte anteriore in modo da affrontare con maggiore facilità l'ostacolo stesso. Durante la scalata, l'accelerometro presente su LionHell controlla l'inclinazione del robot e richiede l'intervento della coda nel caso in cui questa sia richiesta, in particolare quando l'inclinazione è troppo elevata e vi è il rischio che LionHell si ribalti.

Sollevando la parte frontale, LionHell è in grado di adattarsi più facilmente alla conformazione del terreno e ad affrontare con maggiore facilità ogni ostacolo lungo il percorso garantendogli una maggiore flessibilità, come mostrato in altri lavori precedenti [30, 47].

Il sistema di controllo svolge inoltre il ruolo di determinare l'azione da intraprendere nel caso in cui LionHell si appresta ad affrontare un ostacolo, basando la sua decisione sui valori restituiti dai sensori della barra centrale:

- se le distanze rilevate dai tre sensori puntati sul terreno sono simili, allora LionHell procede diritto, considerando il terreno attraversabile;
- se la differenza tra la distanza rilevata dal sensore centrale e il sensore laterale destro o sinistro è alta, allora LionHell ruota a sinistra nel

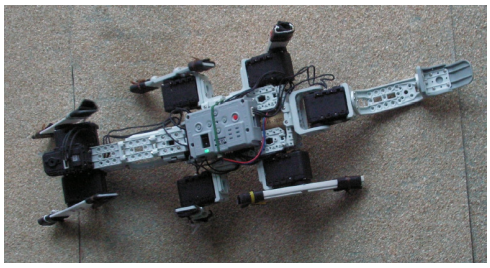
caso in cui la distanza rilevata dal sensore sinistro sia inferiore a quella rilevata dal sensore destro e viceversa nel caso opposto.

LionHell mostra quindi un comportamento diverso in base alla superficie su cui si sta muovendo, dimostrando di essere in grado di adattarsi all'ambiente circostante e di affrontare dinamicamente gli ostacoli che gli si presentano [24, 31, 28].

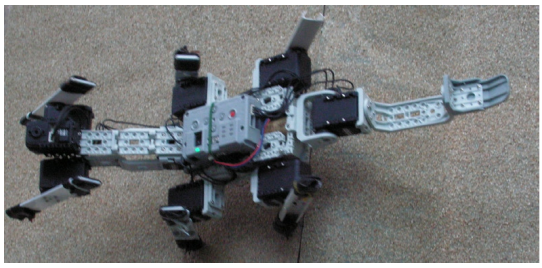
Il risultato è un robot che è in grado di scalare gli ostacoli, ma che non è in grado di curvare, in quanto la barra sensoriale risulta comunque insufficiente per decidere la direzione da prendere nel caso sia presente un ostacolo (per maggiori informazioni al riguardo, si faccia riferimento al Capitolo 3, Sezione 3.2), i Wheg presenti sono inadeguati per un movimento fluido e si rovinano facilmente, la coda non è in grado di sviluppare la forza necessaria per bloccare il robot a terra e in alcuni punti la struttura del robot non è adatta, non essendoci ad esempio una protezione per la scheda di controllo e la barra sensoriale risulta poco resistente.

Figura 2.14: LionHell: sviluppo

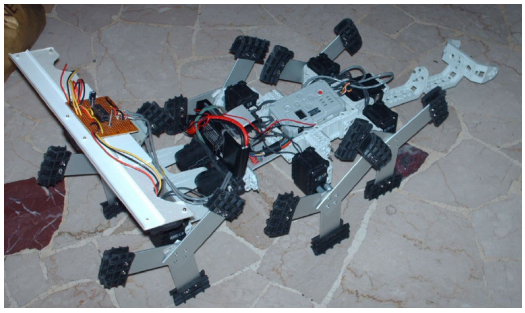
(a) Prima implementazione - A



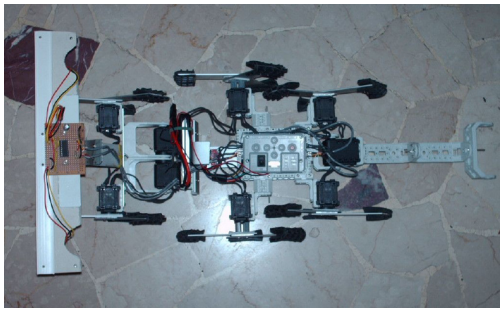
(b) Prima implementazione - B



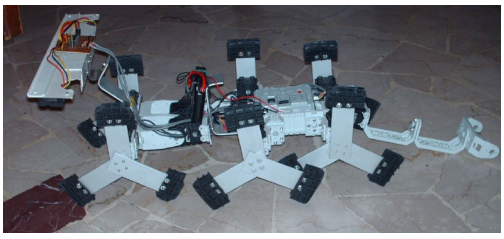
(c) Progetto finale di Vittorio Lumare - A



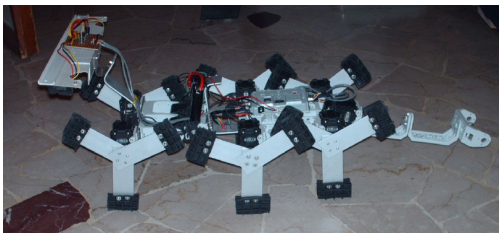
(d) Progetto finale di Vittorio Lumare - B



(e) Progetto finale di Vittorio Lumare - C



(f) Progetto finale di Vittorio Lumare - D



Capitolo 3

LionHell II

LionHell si presenta come un robot mobile, capace di affrontare diversi ostacoli e in grado di muoversi anche su terreni accidentati e sconnessi. Il robot è stato costruito in modo tale da ispirarsi al mondo animale, utilizzando una coda che lo supporta durante le scalate, che deriva dal geco dalla coda larga, e una barra sensoriale frontale in grado di analizzare il terreno, che prende spunto dal funzionamento delle vibrisse del gatto. La mancanza, però, di un sistema sensoriale adeguato ad esplorare ed analizzare il terreno circostante, la struttura talvolta debole in alcuni suoi punti e di fatto, l'incapacità di LionHell di muoversi ci ha spinto a rivalutare LionHell, nel tentativo di migliorare il robot, mantenendone le caratteristiche generali e permettendogli di esplorare l'ambiente anche in mancanza di una mappa della zona.

LionHell II nasce quindi come un progetto di un robot esplorativo esapode, in grado di utilizzare i Wheg come sistema di locomozione, dotato di coda per scalare e di un sistema di telecontrollo remoto che gli permetta di muoversi nell'ambiente circostante. In Figura 3.1 e in Figura 3.2 è possibile osservare LionHell II completo dopo le modifiche effettuate, con il corrispettivo telecomando per il controllo in remoto.

LionHell ha subito una serie di modifiche volte a rinforzare e migliorare il sistema preesistente e a migliorarne l'usabilità e le capacità di movimento. Per cercare di raggiungere tale obiettivo, è stato necessario:

- modificare la meccanica dei Wheg (descritti precedentemente nella Sezione 2.1), rinforzandone la struttura, in quanto i Wheg precedenti tendevano a danneggiarsi facilmente a causa del movimento stesso di LionHell. I nuovi Wheg sono descritti in dettaglio nella Sezione 4.1;

- rinforzare e potenziare la coda, aggiungendo un motore aggiuntivo che permettesse a LionHell di sviluppare la forza necessaria durante le scalate , in particolare nel caso in cui il robot rischiasse di ribaltarsi affrontando un piano molto inclinato, un ostacolo particolarmente alto (>16 cm) rispetto al robot o una serie di ostacoli di altezza media (∼12 cm). La nuova coda è descritta nel Capitolo 4, Sezione 4.3;

- introdurre un sistema che permettesse il telecontrollo del robot da remoto. In questo caso è stata scelta la tecnologia XBee, che permette di interfacciare il corpo di LionHell II con un telecomando controllato in remoto. L'XBee è descritto in dettaglio nella Sezione 3.4, mentre le modifiche riguardanti il telecontrollo che sono state apportate a LionHell sono descritte nel Capitolo 5;

- migliorare l'aspetto visivo generale, sia per rendere il robot appetibile ad un pubblico più vasto, dandogli un aspetto più animale, sia per rinforzare e proteggere parti delicate della componentistica del robot, creando una robusta corazza sul busto. Il design è descritto in dettaglio nel Capitolo 6.

Per poter effettuare tali modifiche, è stato necessario considerare la componentistica preesistente, riguardante gli attuatori che muovono i Wheg, in modo da determinare il movimento di LionHell II in curva,e riguardante i sensori utilizzati nella barra sensoriale, per comprenderne il funzionamento e il ruolo che potrebbero ricoprire nella decisione della direzione da prendere.

La Sezione 3.1 descrive i telemetri che sono stati utilizzati per la barra sensoriale della testa.

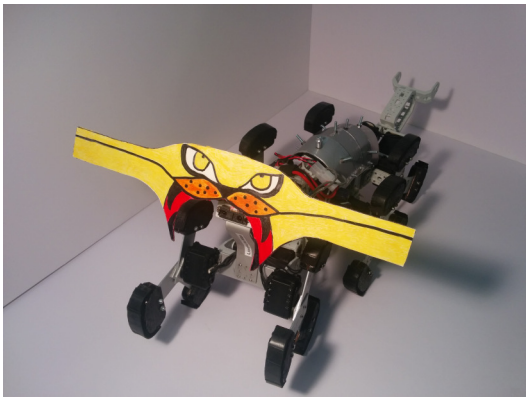
La Sezione 3.2 descrive la barra sensoriale della testa, motivando la ragione che ci ha portato ad utilizzare il controllo remoto.

La Sezione 3.3 descrive il movimento di LionHell II, mostrando i motori che sono stati utilizzati per i Wheg.

La Sezione 3.4 descrive l'XBee, ossia il componente che è stato utilizzato per ottenere il controllo remoto.

Figura 3.1: LionHell Mc Millan II - foto 1

(a) Frontalmente



(b) Telecomando - 1

(c) Lateralmente

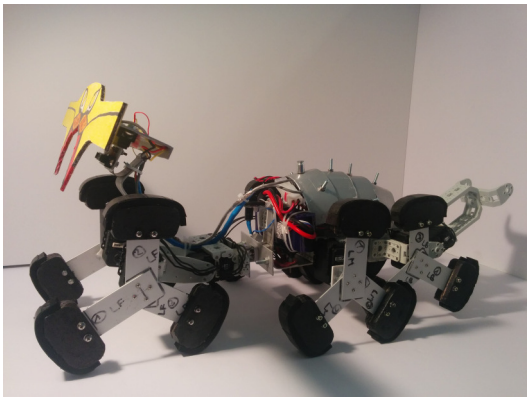
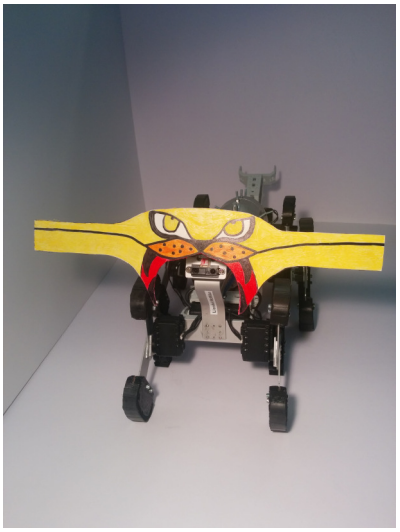
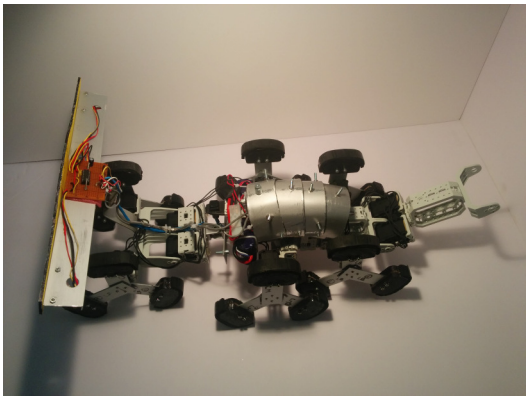


Figura 3.2: LionHell Mc Millan II - foto 2

(a) Volto



(b) Dall'alto

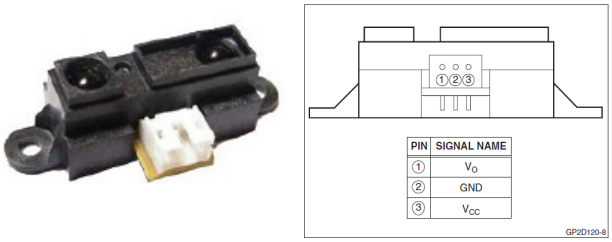


(c) Telecomando - 2

3.1 I telemetri Sharp

Figura 3.3: Il telemetro Sharp GP2D120X

(a) Esterno (b) Schema



LionHell II è dotato di telemetri Sharp del modello GP2D120X, mostrati in Figura 3.3, utilizzati per rilevare se il terreno è sgombro oppure no, individuare la presenza di ostacoli superabili e l'altezza degli stessi. Tali sensori sono installati nella testa, descritta in dettaglio nella Sezione 3.2.

Le specifiche del sensore sono descritte nella Tabella 3.1, mentre la Figura 3.4a mostra la funzione distanza-voltaggio caratteristica dell'output del sensore. Come mostrato in Figura 3.4b, la funzione non è lineare e necessita di una tabella che leghi il valore della tensione alla distanza corrispondente. Infine, in Figura 3.5 si può osservare l'allineamento appropriato del sensore rispetto al moto della superficie, che spiega la ragione che ci ha portato a modificare l'allineamento dei sensori della testa, come mostrato in Figura 3.6.

Tabella 3.1: Specifiche del telemetro Sharp GP2D120X

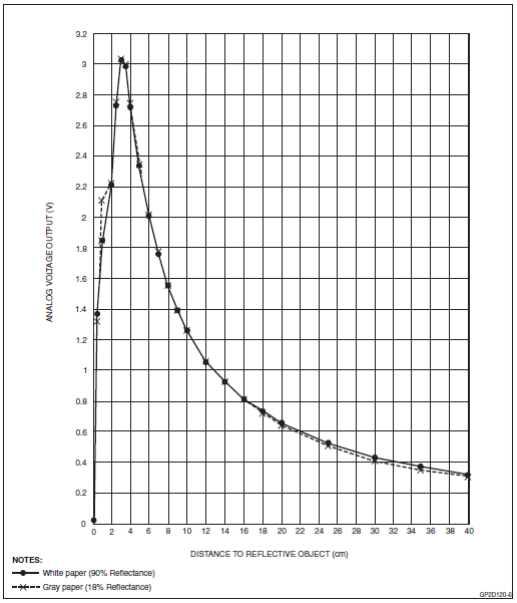
Parametro	Simbolo	Condizioni	Min.	Media	Max.	Unità	Note
Misurazione Gamma di Distanza	$\triangle L$		4	-	30	cm	1, 2
Tensione di Uscita del Terminale	V_o	$L = 30\ cm$	0.25	0.4	0.55	V	1, 2
Differenza della Tensione di Uscita	$\triangle V_o$	$30\ cm \geq \triangle L \geq 4\ cm$	1.95	2.25	2.55	V	1, 2
Corrente di Alimentazione Media	I_{CC}	$L = 30\ cm$	-	33	50	mA	1, 2
Tensione di Esercizio	V_{CC}		4.5	-	5.5	V	

1. Misure effettuate tramite Kodak R-27 Gray Card, usando il lato bianco (riflettività del 90%)

2. L = distanza dall'oggetto riflesso

Figura 3.4: Funzioni del telemetro Sharp GP2D120X

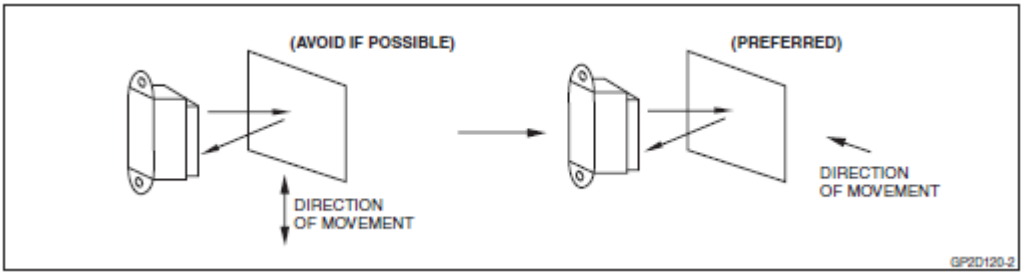
(a) Diagramma



(b) Codice

```
unsigned short sharp_calib[17][2] =
{
    {63,40},
    {71,35},
    {85,30},
    {106,25},
    {132,20},
    {153,18},
    {165,16},
    {192,14},
    {214,12},
    {257,10},
    {286,9},
    {319,8},
    {360,7},
    {415,6},
    {480,5},
    {562,4},
    {613,3}
}
```

Figura 3.5: Allineamento appropriato per superfici in movimento



3.2 Il sistema di percezione sensoriale della testa

LionHell è in grado di percepire l'ambiente esterno e la presenza di ostacoli di fronte a sé grazie all'utilizzo di una barra sensoriale posizionata sulla testa, come mostrata in Figura 3.6b. I sensori presenti sono telemetri Sharp GP2D120X, descritti nella Sezione precedente 3.1, e sono posizionati in modo tale da poter coprire la larghezza di LionHell:

- Un sensore è posizionato al centro della barra, rivolto in avanti, il suo ruolo è quello di individuare la presenza di possibili ostacoli;
- Un altro sensore è posizionato al centro della barra, rivolto verso il basso, il suo ruolo è quello di analizzare il terreno di fronte a sé;
- due sensori sono posizionati lateralmente, e svolgono il ruolo analizzare la superficie del terreno come nel caso del sensore centrale precedentemente descritto.

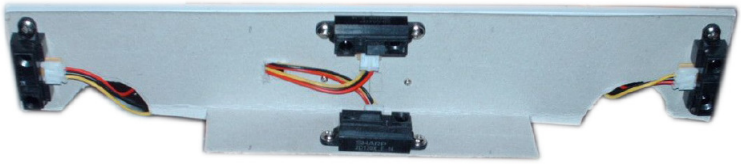
Il progetto iniziale prevedeva l'utilizzo dei sensori della barra centrale per poter discriminare la direzione da prendere nel caso in cui il robot avesse dovuto affrontare un ostacolo:

- se le distanze rilevate dai tre sensori puntati sul terreno fossero state tutte simili, considerando una tolleranza minima, allora il robot avrebbe considerato il terreno attraversabile;
- se la differenza tra la distanza rilevata dal sensore centrale e il sensore laterale destro o sinistro fosse stata molto alta, allora LionHell avrebbe girato a sinistra nel caso in cui la distanza rilevata dal sensore sinistro fosse stata inferiore a quella rilevata dal sensore destro e viceversa nel caso opposto;

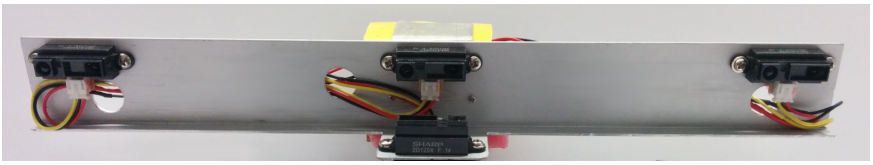
I sensori presenti, ciononostante, sono troppo pochi e troppo limitati per poter prendere vere e proprie decisioni e non sono in grado di analizzare

Figura 3.6: Il sistema sensoriale della testa

(a) LionHell: Barra sensoriale originale



(b) LionHell II: Barra sensoriale nuova



sufficientemente in dettaglio l'ambiente circostante per poter comprendere quale sia la strada migliore da scegliere. Ad esempio, LionHell è capace di comprendere se un ostacolo è superabile oppure no, ma nel caso dell'esempio in Figura 3.7 LionHell si ritroverebbe a dover gestire le seguenti informazioni:

1. il sensore frontale indica la presenza di un ostacolo superabile: il robot deve decidere se affrontarlo od evitarlo;

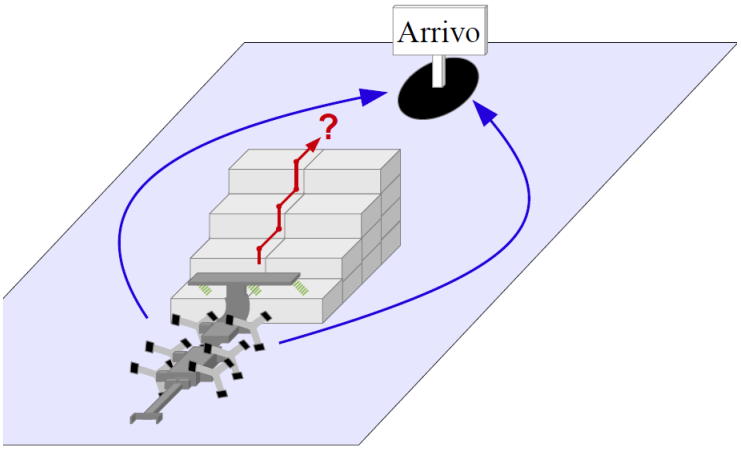
2. i tre sensori che puntano sul terreno indicano la stessa distanza approssimativa dalla superficie, ossia la parte superiore del primo gradino (evidenziato in Figura 3.7 dalle spesse linee verdi tratteggiate);

3. essendo l'ostacolo superabile e non rilevando differenze di distanze tra i sensori laterali e quello centrale, LionHell decide di proseguire dritto (evidenziato in Figura 3.7 dalla freccia rossa);

4. sulla cima dell'ostacolo, LionHell controlla se può scendere, ma l'altezza del salto risulta troppo elevata ed il robot rinuncia a raggiungere il suo obiettivo.

Nel caso in cui LionHell avesse avuto un sistema di sensori più complesso, sarebbe stato in grado di osservare quello che lo circondava e probabilmente avrebbe scelto uno dei due percorsi evidenziati dalle frecce blu. Inoltre, a

Figura 3.7: LionHell: dritto, destra o sinistra?



causa dei continui sobbalzi provocati dal movimento dei Wheg, la testa tende ad inclinarsi a destra e a sinistra, compromettendo i valori letti dai sensori e causando la rilevazione di ostacoli inesistenti.

Da questo semplice esempio è facile intuire che il sistema sensoriale di cui LionHell è dotato risulta insufficiente per esplorare l'ambiente circostante ed essere in grado di effettuare scelte consapevoli sulla strada migliore da percorrere per raggiungere un obiettivo. A questo punto sorge spontanea una domanda: è possibile migliorare il sistema sensoriale affinché LionHell II sia in grado di muoversi autonomamente, ad esempio aggiungendo una coppia di telecamere, oppure ricorrere direttamente ad un controllo remoto, in modo da poter decidere personalmente la direzione da prendere? Cercheremo di rispondere a questa domanda considerando che cosa comporta avere una coppia di telecamere sulla testa e quali sono i vantaggi dal possedere invece un telecomando ed averne quindi un controllo diretto.

L'utilizzo di due telecamere frontali comporterebbe un aumento totale del peso della testa di LionHell II, il che non ne faciliterebbe certamente la capacità di superare ostacoli di medie o grandi dimensioni. Inoltre, il robot rischierebbe di danneggiarle durante il movimento, in particolare quando si appresta a scendere da ostacoli di grandi dimensioni c'è il rischio che colpisca frontalmente le telecamere. Oltre a questo, è necessario che le telecamere siano

efficacemente precise in modo da ignorare i sobbalzi a cui il robot può andare incontro a causa di irregolarità del terreno e durante le scalate, permettendogli di seguire sempre il percorso ottimale al fine di raggiungere l'obiettivo prefissato, il che richiede anche un controllo dedicato delle telecamere, in quanto la scheda di controllo CM-510 sarebbe insufficiente.

Con l'utilizzo di un telecomando, al contrario, si è in grado di controllare completamente il robot e il controllo remoto può essere installato sul busto anziché sulla testa, mantenendola leggera e permettendole di scalare gli ostacoli senza ulteriori pesi. Inoltre, tale controllo sarebbe totalmente integrato all'interno di una scheda di piccole dimensioni chiamata XBee, installata direttamente sopra la scheda di controllo. L'utilizzo di un telecomando permette di esplorare l'ambiente circostante con facilità, mentre il controllo interno alla scheda gestisce indipendentemente le azioni che devono essere svolte per scalare un ostacolo di piccole, medie o grandi dimensioni, modificando la forza richiesta dai motori, alzando di molto o di poco la testa e azionando la coda in caso di necessità, ed interrompendone il movimento nel caso in cui LionHell II si dovesse trovare a dover superare un salto di altezza troppo elevata.

Utilizzando il telecontrollo, il risultato è un robot dotato sia di comportamenti autonomi, sia di un sistema di controllo in remoto che permette all'utente di decidere il percorso desiderato, se procedere dritto, fermarsi o curvare a destra o a sinistra. Il controllo autonomo interviene nel momento in cui LionHell II si appresta ad affrontare un ostacolo: il sensore frontale individua la presenza dell'ostacolo in questione e il sistema di controllo solleva la parte anteriore del robot per facilitarlo durante la scalata e attiva la coda nel caso in cui l'ostacolo sia alto, e in generale quando vi è il rischio che il robot possa ribaltarsi a causa dell'elevata inclinazione.

Il robot mantiene dunque una serie di comportamenti autonomi nel momento in cui deve superare un ostacolo, mentre in parte è anche controllato dall'utente, il quale deve decidere la direzione che il robot deve prendere. In questo modo il robot è in grado di mostrare una serie di comportamenti dinamici basati sulla lettura dei dati dei sensori (percezione), la loro modellizzazio-

ne, la pianificazione delle successive azioni da intraprendere, l'esecuzione del compito e infine il controllo diretto dei motori [23]. Il controllo remoto combinato ai comportamenti dinamici del robot permette a LionHell II di adattarsi attivamente all'ambiente circostante, rispondendo anche a casi particolari in cui il solo controllo autonomo risulta insufficiente.

3.3 Il movimento di LionHell II

LionHell II basa il suo movimento sull'utilizzo di 10 attuatori Dynamixel AX-12 [49, 50], mostrati in Figura 3.8, che controllano, in modalità differenti, I Wheg presenti, la testa e la coda di LionHell II.

Gli attuatori possono funzionare in due differenti modalità [13]:

- modalità di rotazione continua, gli attuatori svolgono il ruolo di ruote, o Wheg, come in questo caso;
- modalità di posizione, permettono il controllo della coda e del giunto centrale.

Gli attuatori causano alcuni problemi solo nel momento in cui lavorano nella banda morta a 60° (come mostrato in Figura 3.9). Quando si trovano a lavorare in questa fascia, il potenziometro non è in grado di individuare la posizione del servo. Per questa ragione non è possibile sincronizzare perfettamente tutti i motori nell'istante in cui LionHell II si appresta ad affrontare l'ostacolo: il controllo dei motori consiste dunque in un loop aperto.

In LionHell II, 6 attuatori in modalità continua servono per controllare i sei Wheg, il che significa che ogni Wheg è indipendente dagli altri, mentre gli altri 4 si suddividono in due per il controllo del giunto centrale e due per il controllo della coda, arrivando in totale all'utilizzo di 10 attuatori.

Sfruttando il fatto che gli attuatori possano intervenire e ruotare indipendentemente l'uno dall'altro, è stato possibile fare in modo che LionHell II curvasse sfruttando una differenza di velocità tra i Wheg, e permettendogli

Figura 3.8: Dynamixel AX-12

(a) Esterno (b) Schema

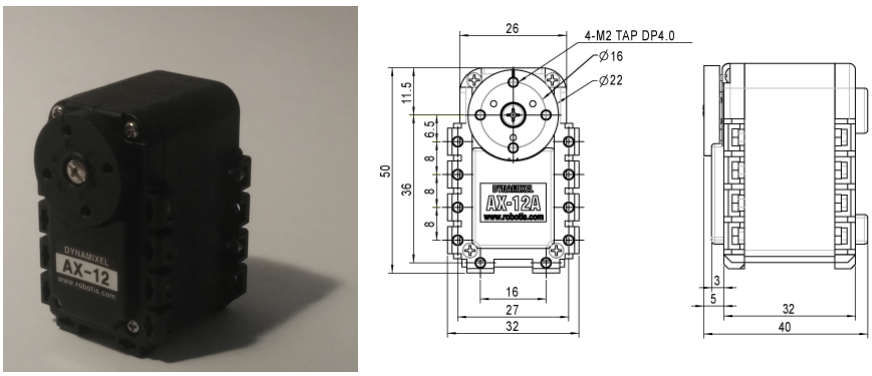


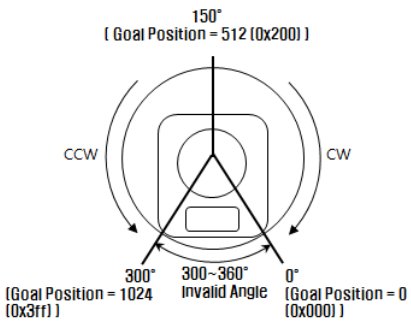
Tabella 3.2: Specifiche del Dynamixel AX-12

Parametro	Valore	Unità	Note
Peso	53.5	g	
Risoluzione	0.29	°	
Tasso di Riduzione	254:1	-	
Tensione di Ingresso	$9 \sim 12$(raccomandato 11.1)	V	
Coppia di Stallo	1.5	N, m	1
Velocità a Vuoto	59	rpm	2

1. Misurazione effettuata a $12,0V$, $1.5A$

2. Misurazione effettuata a $12.0V$

Figura 3.9: Banda morta nel Dynamixel AX-12



anche di ruotare su se stesso sfruttando i motori indipendenti di cui possiede. Affinché ciò fosse possibile, è stato necessario creare un sistema di controllo remoto che fosse in grado di comunicare con la scheda di controllo in modo da ricevere ed interpretare i comandi ricevuti dal telecomando. A questo scopo è stata utilizzata una scheda XBee (descritta nella Sezione 3.4, Datasheet nell'Appendice C)

3.4 XBee

L'XBee [37, 20] è un fattore di forma compatibile con i moduli radio basati su IEEE 802.15.4, ed è una marca della "Digi International". Inizialmente furono introdotti due modelli, un XBee a basso costo da 1 mW ed un altro con maggiore potenza, XBee PRO da 100 mW. Tutti gli XBee possono essere utilizzati con il numero minimo di connessioni, ossia l'alimentazione (3.3 V), la messa a terra, l'ingesso e l'uscita dei dati.

L'XBee scelto è un XBee 4214A da $1mW$, è dotato di 20-pin di ingresso (mostrati in Figura 3.10b e descritti in Tabella 3.3) e di una antenna wire (antenna a filo) che ne permette la comunicazione via radio. L'XBee può operare sia in una modalità dati trasparente sia per mezzo di una interfaccia di programmazione dell'applicazione (API).

La funzionalità usata è chiamata Virtual Wire [18] (filo virtuale) e sfrutta l'utilizzo di onde ad alta frequenza per trasmettere i pacchetti dati. Nelle esistenti architetture di emulazione sia la configurazione logica che la connettività di rete rimangono fisse per tutta la durata dell'emulazione (Physical Wire, ossia filo fisico). Ogni partizione emulata consiste in un insieme di porte e di segnali che comunicano con le altre partizioni. Ciascuna porta emulata è mappata ad una o più porte FPGA (Field Programmable Gate Array, si tratta di un circuito integrato caratterizzato da elevata scalabilità le cui funzionalità sono disponibili via software, consente l'implementazione di funzioni logiche equivalenti anche molto complesse) e ciascun segnale emulato inter-partizione è allocato in una coppia di pin tra due FPGA. Quindi, affinché una partizione sia possibile, è necessario che i requisiti di pin e porte

Figura 3.10: XBee

(a) XBee: fronte (b) XBee: schema (c) XBee: retro

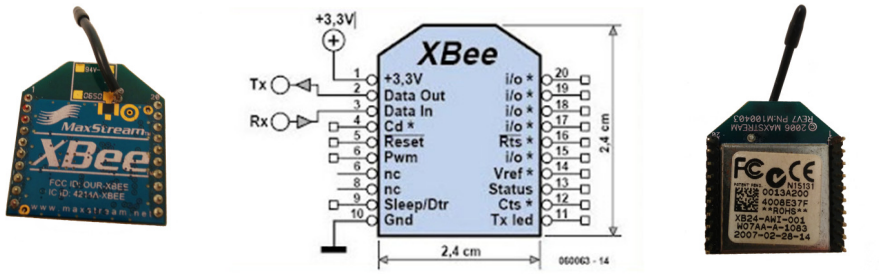
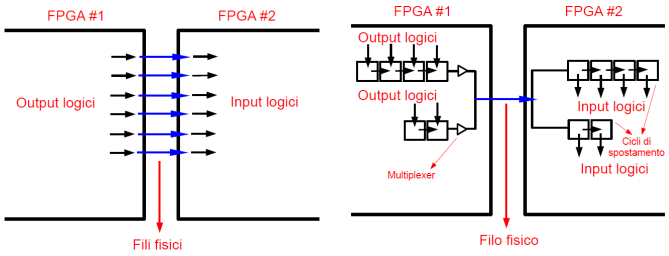


Figura 3.11: Physical Wire e Virtual Wire

(a) Physical Wire (b) Virtual Wire



non sia superiore alle risorse disponibili di un FPGA. Il Virtual Wire elimina il problema di limitazione dei pin multiplando intelligentemente ciascun filo fisico attraverso diversi fili logici multipli eseguendo in pipeline tali connessioni alla frequenza massima dell'FPGA. Un Virtual Wire rappresenta una semplice connessione tra l'output logico di un FPGA e l'input di un altro FPGA e l'utilizzo di tale tecnologia permette di incrementare la banda di comunicazione multiplando l'uso delle risorse pin FPGA (fili fisici) tra più segnali di emulazione (fili logici).

In Figura 3.11 è possibile osservare la differenza tra le due tecnologie: in Physical Wire (Figura 3.11a) ad ogni entrata ed uscita dell'FPGA corrisponde uno ed uno solo filo fisico, che connette direttamente gli FPGA, mentre nel caso del Virtual Wire (Figura 3.11b) esiste un solo filo fisico, e tutte le connessioni sono effettuate per mezzo di un multiplexer che multipla ogni filo fisico attraverso diversi fili logici, come precedentemente descritto.

Tabella 3.3: XBee: pedinatura

Pin	Nome	Direzione	Descrizione	AT	PR
1	VCC	-	Alimentazione elettrica		
2	$DOUT$	Output	UART dati in uscita		
3	$DIN/\overline{CONFIG}$	Input	UART dati in ingresso		7
4	$DO8$	Output	Output digitale 8		
5	$\overline{RESET}$	Input	Modulo di reset (almeno $200nS$)		
6	$PWM0/RSSI$	Output	PWM output 0 / Ind. della forza del segnale RX		
7	$PWM1$	Output	Output PWM 1		
8	(riservato)		Non connesso		
9	$\overline{DTR}/SLEEP\_RQ/D18$	Input	Pin di controllo SLEEP o Input digitale 8	D8	6
10	GND	-	Messa a terra		
11	$AD4/DIO4$	Output o Input	Input A 4 o I/O digitale 4	D4	0
12	$\overline{CTS}/DIO7$	Output o Input	Controllo del flusso o I/O digitale 7	D7	
013	$ON/\overline{SLEEP}$	Output	Indicatore dello status del modulo		
14	$VREF$	Input	Riferimento di tensione per gli Input AD		
15	$associato/AD5/DIO5$	Output o Input	Ind. associato, Input analogico 5 o I/O digitale 5	D5	
16	$RTS/AD6/DIO6$	Output o Input	Contr. flusso RTS, Input analogico 6 o I/O digitale 6	D6	5
17	$AD3/DIO3$	Output o Input	Input analogico 3 o I/O digitale 3	D3	1
18	$AD2/DIO2$	Output o Input	Input analogico 2 o I/O digitale 2	D2	2
19	$ADI/DIO1$	Output o Input	Input analogico 1 o I/O digitale 1	D1	3
20	$AD0/DIO0$	Output o Input	Input analogico 0 o I/O digitale 0	D0	4

1. La colonna AT fornisce la coda del comando Dn per configurare il pin

2. La colonna PR fornisce il numero dei bit nel comando PR per configurare i resistori pull-up

3. UART significa ricevitore-trasmettitore asincrono universale, è un dispositivo hardware di uso generale o dedicato, converte i flussi di bit da un formato parallelo a un formato seriale asincrono o viceversa

4. PWM è la modulazione di larghezza di impulso, permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e quello negativo (ciclo di lavoro), è utilizzata per i protocolli di comunicazione in cui l'informazione è codificata sotto forma di durata del tempo di ciascun impulso. In questo caso, il periodo è di $64\mu S$, e vi sono 1023 (0x3ff) passi in essa, con una frequenza di $15.6KHz$ e un ciclo di lavoro che varia in 1024 passi tra lo 0% e il 100%

5. RX è il ricevitore radio, è adibito a ricevere segnali informativi ad esso in input provenienti dal canale di comunicazione

6. La linea RTS viene utilizzata dall'Host, per segnalare al XBee che il buffer dell'Host è quasi pieno e l'XBee dovrebbe interrompere la trasmissione fino a quando la linea non è più richiesta.

Capitolo 4

Controllo e Mobilità di LionHell II

In questo capitolo descriveremo le modifiche prevalentemente meccaniche e strutturali che sono state effettuate su LionHell, esponendo le motivazioni delle nostre scelte.

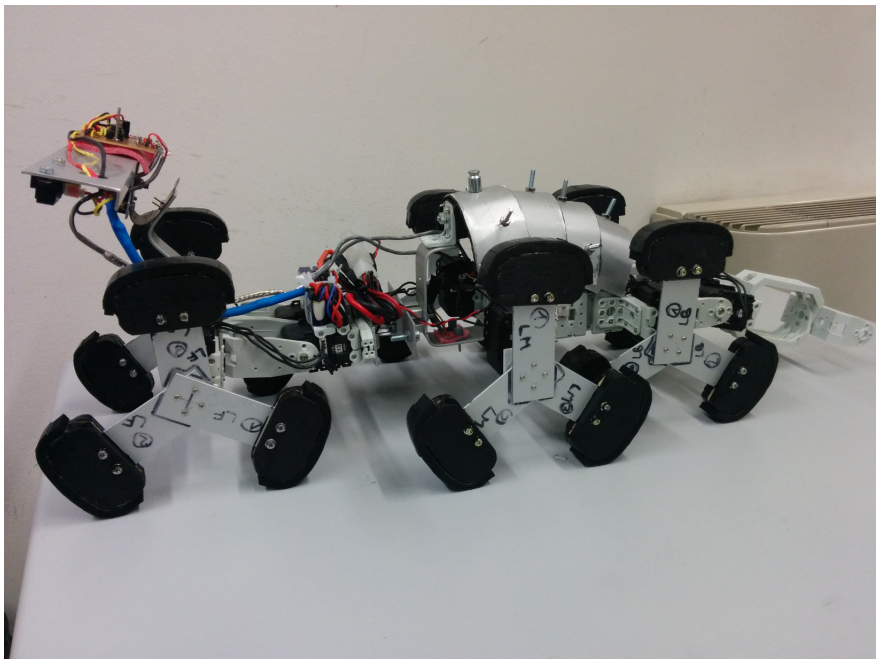
La Sezione 4.1 descrive i nuovi Wheg utilizzati su LionHell II e le modifiche che sono state effettuate rispetto ai Wheg originali.

La Sezione 4.2 descrive il giunto passivo centrale in LionHell II e la sua funzione.

La Sezione 4.3 descrive il ruolo della coda in LionHell II e negli animali, in particolare nel geco.

4.1 Wheg in LionHell II

Figura 4.1: LionHell II: struttura esterna



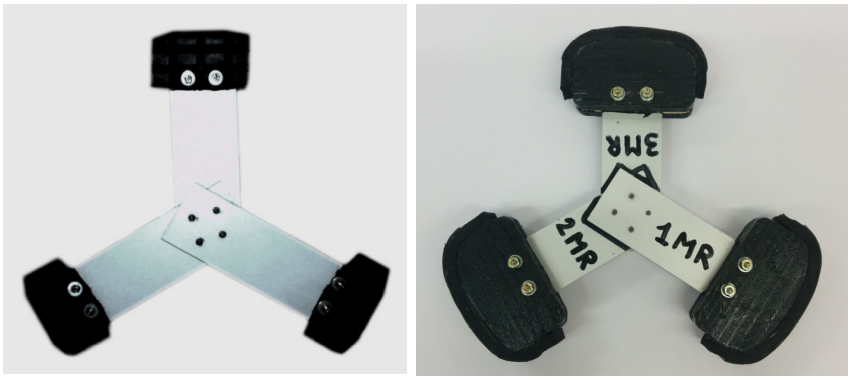
LionHell II possiede in totale 6 Wheg, come nel caso di Rhex (Figura 4.1), ma ciascun Wheg è composto da 3 barre disposte a 120° di distanza le une dalle altre, alla cui estremità è stato montato un piede leggermente curvo in modo da garantire una presa sicura sul terreno. Il movimento dei Wheg è simultaneo e ciascun Wheg è controllato da un motore indipendente che lavora in modalità continua, garantendo al robot un movimento fluido e adatto ad ogni situazione.

Il Wheg che LionHell II possiede è caratterizzato da:

- una struttura basale, in alluminio, garantisce il punto di contatto con lo snodo centrale;
- una struttura basale del piede, in legno, modellato secondo una forma curva in modo da adattarsi meglio al terreno ed evitare sobbalzi;

Figura 4.2: Confronto tra Wheg

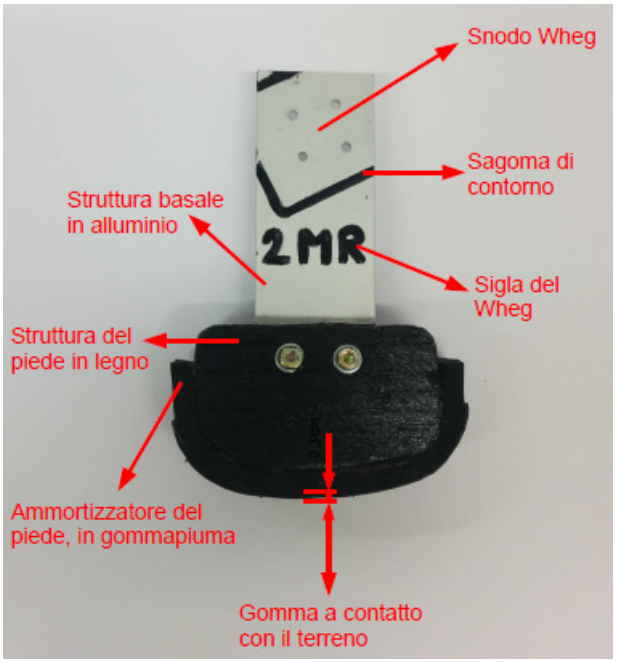
(a) LionHell: Wheg originale (b) LionHell II: Wheg rinforzato



- la gommapiuma, a contatto con la struttura del piede, riduce gli urti del robot a contatto con il terreno;
- la gomma, a diretto contatto con il terreno, protegge la gommapiuma dall'usura e permette al robot di muoversi agevolmente su qualunque superficie;
- la sigla permette di identificare con facilità i componenti del Wheg e la posizione della gamba stessa nel robot;
- la sagoma di contorno permette di capire intuitivamente la posizione con cui gli altri componenti del Wheg vanno montati .

Rispetto al precedente modello, abbiamo mantenuto l'asta di metallo base e rimosso la gomma che faceva da piede, in quanto il movimento stesso di LionHell avrebbe danneggiato irrimediabilmente i Wheg ed in particolare il delicato punto di appoggio.

Figura 4.3: LionHell II: Wheg rinforzato, particolare



4.2 Giunto passivo centrale

In questa sezione descriveremo il ruolo del nuovo giunto centrale passivo e l'incremento dei gradi di libertà che ne ha comportato.

La Sottosezione 4.2.1 definisce il termine Gradi di libertà e dimostra che ne sono richiesti almeno sei per ogni robot per risultare versatile.

La Sottosezione 4.2.2 mostra i Gradi di libertà di LionHell II, considerando tutti gli attuatori presenti.

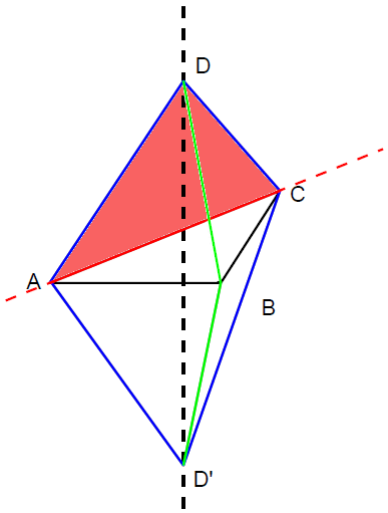
La Sottosezione 4.2.3 mostra il giunto centrale passivo che è stato aggiunto e il ruolo dello stesso.

4.2.1 Gradi di libertà: definizione

Il numero di gradi di libertà (DoF, ossia Degree of Freedom) di un punto materiale è il numero di variabili indipendenti necessarie per determinare univocamente la sua posizione nello spazio (coordinate).

I Gradi di libertà sono un termine utilizzato per definire la libertà di movimento di un robot nelle tre abilità spaziali, e il numero di gradi di libertà di un robot ne definiscono la configurazione.

Figura 4.4: Gradi di libertà di un corpo rigido



In particolare, si può dimostrare [56] che un robot necessita di almeno sei gradi di libertà per essere completamente versatile. Come esempio, si consideri un corpo rigido: per determinarne la posizione nello spazio è sufficiente conoscere la posizione in tre punti non allineati A, B e C. Ogni altro punto D è infatti determinabile considerando il triangolo ACD (Figura 4.4, il triangolo in rosso): la base AC è fissata e D ha una distanza fissa da A e da C (sia AD che DC sono segmenti blu). Ruotando il triangolo ACD intorno all'asse passante per AC (linea tratteggiata rossa), è possibile determinare la posizione di D' che si trova alla stessa distanza di D da B (D'B e DB sono due linee verdi). Come si può osservare in figura, D' si trova dalla parte opposta rispetto al piano ABC, quindi esiste un solo punto D che abbia una distanza fissata da A, B e C e che si trovi da un lato fissato del piano ABC.

Il sistema di punti ABC ha $9 - f$ gradi di libertà (3 gradi di libertà per ogni punto, ossia le tre coordinate necessarie per determinare la posizione di un punto nello spazio), dove f è il numero di vincoli. Essendo che le distanze

AB, BC e AC devono rimanere costanti, ne consegue che $f = 3$ e quindi il corpo ha in totale 6 gradi di libertà.

4.2.2 Gradi di libertà di LionHell II

LionHell II è un robot esapode dotato di Wheg, coda, un giunto motorizzato che gli permette di sollevare la parte frontale per poter affrontare meglio gli ostacoli, ed infine di un giunto centrale passivo. In Figura 4.5 è possibile osservare i gradi di libertà di LionHell II:

- la Figura 4.5a mostra il movimento dei Wheg: ciascun Wheg è dotato di 1 grado di libertà, per un totale di 6 gradi di libertà;
- la Figura 4.5b mostra il movimento del giunto motorizzato e della coda, ciascuno dei quali aggiunge 1 grado di libertà, per un totale di 2 gradi di libertà;
- la Figura 4.5c mostra invece il giunto centrale passivo che è stato aggiunto successivamente, e le linee verdi tracciano un possibile movimento della parte frontale del robot.

La creazione di un robot con un giunto centrale passivo in più è nata dall'idea di facilitarne i movimenti durante le curve e permettergli un movimento più fluido. Nella sottosezione successiva spiegheremo più dettagliatamente il giunto che è stato aggiunto.

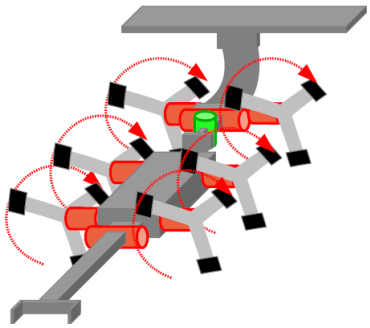
4.2.3 Giunto passivo centrale in LionHell II

LionHell II è dotato di un giunto passivo centrale, il cui ruolo è quello di accompagnare il movimento dei Wheg facilitandone il movimento quando il robot si appresta a curvare. Il giunto passivo è costituito da:

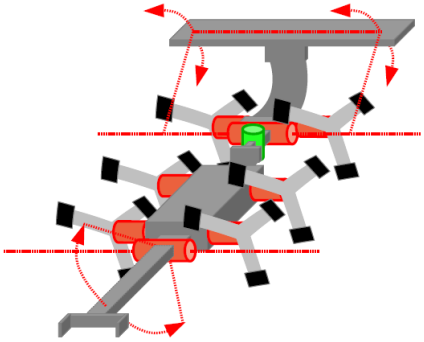
- un organo di collegamento che permette la rotazione di una parte del corpo di LionHell II;

Figura 4.5: Gradi di libertà di LionHell II

(a) Wheg



(b) Giunto centrale motorizzato e Coda



(c) Giunto centrale passivo

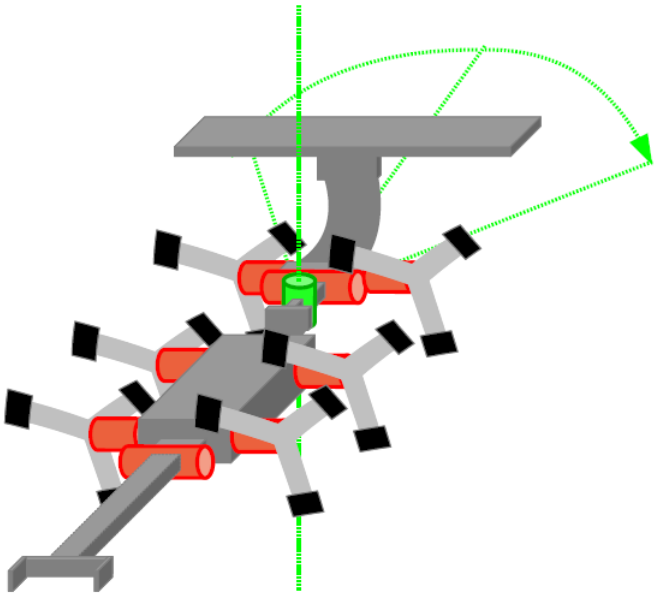
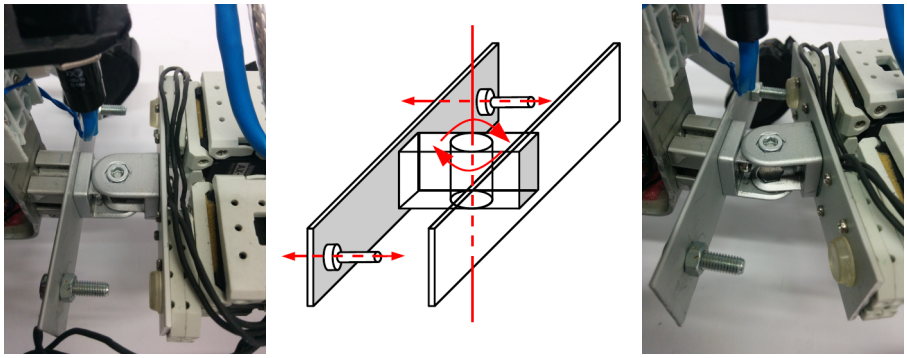


Figura 4.6: Giunto passivo centrale

(a) Dall'alto (b) Schema (c) Lateralmente



- una coppia di barre metalliche alle due estremità dell'organo di collegamento, fanno parte del sistema di sicurezza;

- due viti alle estremità delle barre, il loro ruolo è quello di delimitare l'angolo di rotazione massima del giunto.

Le viti presenti possono essere sostituite con viti più corte o più lunghe, a differenza che si voglia un angolo di rotazione inferiore o superiore. Nello stato attuale, il giunto è capace di ruotare di $\sim 10°$ a destra o a sinistra, mentre con l'assenza totale delle viti l'angolo aumenta fino a $\sim 30°$. L'aggiunta delle viti si è resa necessaria in quanto si era presentato il rischio che i Wheg anteriori venissero a collidere con i Wheg intermedi, rischiando di bloccare il robot nelle curve più strette.

L'aggiunta del giunto passivo si è resa necessaria a causa della lunghezza di LionHell e della difficoltà che questo aveva nell'effettuare alcune curve strette, e l'idea di base era quella di simulare, sotto certi aspetti, il gancio presente nei rimorchi e nei treni, con la differenza che in questo caso anche il rimorchio è in grado di curvare, facilitandone quindi il movimento.

4.3 Coda

In questa sezione descriveremo il ruolo della coda nel robot LionHell II, descrivendo il motivo che ci ha portato a mantenere la coda del progetto originale, ma aumentandone la robustezza ed aumentando la potenza dei motori.

Nella Sottosezione 4.3.1 ci dedicheremo in particolare al ruolo che la coda svolge negli animali, dimostrandone la sua efficacia nel superare ostacoli di grandi dimensioni.

Nella Sottosezione 4.3.2 mostreremo la nuova coda di LionHell II e le modifiche che sono state apportate rispetto alla coda del progetto originale.

4.3.1 Ruolo della coda negli animali

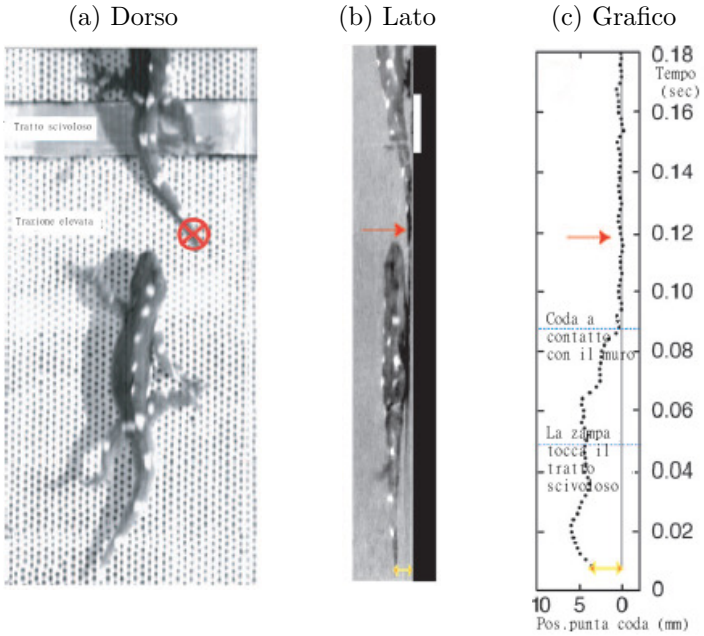
La coda è un elemento fondamentale in molti animali e svolge un ruolo attivo quando l'animale si appresta a scavalcare ostacoli di grandi dimensioni [17, 27, 40], o quando deve superare superfici lisce.

Per esaminare tale comportamento è stato scelto il geco Cosymbotus Platyurus, in quanto estremamente agile e con una coda piatta e larga molto attiva. I risultati [17] dimostrano che la coda non svolge solamente il ruolo di struttura passiva che immagazzina il grasso, garantisce l'equilibrio e un appiglio, ma agisce anche come quinta gamba di emergenza durante scalate molto rapide e su superfici scivolose.

In natura degli scalatori rapidi devono rispondere a continui cambiamenti del terreno, la presenza di ostacoli, supporti discontinui e superfici scivolose. Al geco sono state sottoposte due prove, due differenti superfici da scalare caratterizzate da differenti gradi di attrito.

Per prima cosa [17] è stata scelta una pista con una superficie di aderenza elevata, ottenuta da una scheda perforata. I gechi che hanno affrontato la sfida hanno dimostrato di possedere un'elevata aderenza ed equilibrio, e la coda si manteneva a distanza dalla parete fintantoché le zampe mantenevano una presa sicura. Successivamente, è stata aggiunta una striscia scivolosa: in risposta, l'avampiede del geco è scivolato verso il corpo e lo slittamento della

Figura 4.7: Il Geco Cosymbotus Platyurus mentre affronta la 1^a sfida

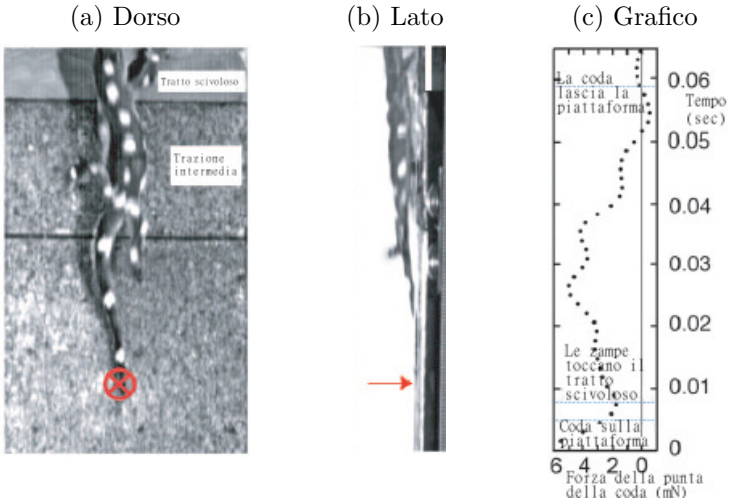


zampa ha provocato la risposta della coda in modo da compensare la perdita dell'appiglio. I gechi dell'esperimento che hanno affrontato la sfida hanno iniziato a muovere la coda contro il muro $\approx 28.9 \pm 6.3ms$ dopo che hanno perso la presa sulla superficie. L'elevatissima velocità di reazione suggerisce che si tratti di un riflesso incondizionato.

In Figura 4.7 è possibile osservare il geco mentre affronta la prima sfida. La Figura 4.7a mostra la visione dorsale del geco mentre affronta sia il tratto con trazione elevata, sia la striscia scivolosa. La Figura 4.7b dimostra che la coda rimane distante dalla superficie fintantoché il geco mantiene una presa solida, ma entra in contatto con la parete subito dopo che la zampa inizia a scivolare. La Figura 4.7c mostra la reazione della coda in funzione del tempo e della distanza della punta della coda dalla parete.

Per testare la nuova ipotesi, è stata cambiata la superficie di contatto [17], scegliendone una con trazione intermedia. A differenza del caso precedente, il geco che corre su tale superficie mantiene la coda a contatto con la parete per tutto il tempo della scalata. A questo punto è stata aggiunta la striscia

Figura 4.8: Il Geco Cosymbotus Platyurus mentre affronta la 2^a sfida

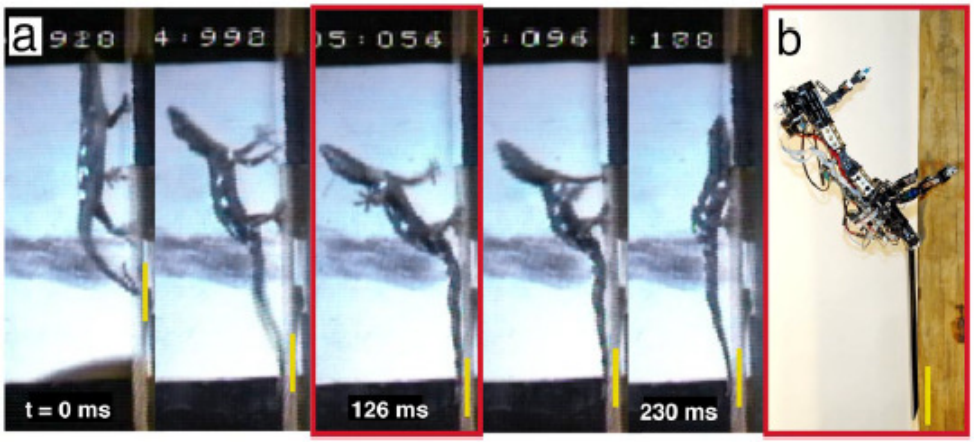


scivolosa come in precedenza ed i calcoli hanno dimostrato che la risposta della coda controbilancia la perdita dell'appoggio sulla superficie.

In Figura 4.8 Il geco affronta la seconda sfida, dove la superficie del percorso è stata sostituita con una con aderenza intermedia, con un sottostrato costituito da una pedana di forza. La Figura 4.8a mostra il dorso del geco mentre si arrampica, la Figura 4.8b ne mostra il lato, mentre la croce e la freccia rosse mostrano il punto di contatto della coda con la parete dopo che le zampe iniziano a scivolare. La Figura 4.8c mostra la forza normale sviluppata dalla punta della coda in funzione del tempo.

La risposta della coda potrebbe non essere in grado di correggere il movimento del geco per strisce scivolose larghe e ripetute più volte. Quando la risposta della coda risulta insufficiente [17], il geco è disposto anche a subire un piegamento all'indietro di circa 60°, eventualmente prevenendone la caduta posizionando la coda in una postura dove gli ultimi due terzi della coda premono contro il muro similarmente al cavalletto di una bicicletta. Anche in caso di queste perturbazioni estreme, il geco non cade mai dal muro, mentre lo stesso esperimento effettuato con altri animali privi di coda ne ha causato la caduta nel 20% dei casi. Inoltre, non è stato individuata una variazione della velocità di movimento tra animali con la coda e animali che ne erano

Figura 4.9: Il Geco Cosymbotus Platyurus mentre si piega all'indietro



privi e in più del 60% dei casi gli animali privi di coda hanno fallito nell'attraversare il tratto scivoloso, mentre negli animali con la coda la percentuale scende al di sotto del 15%.

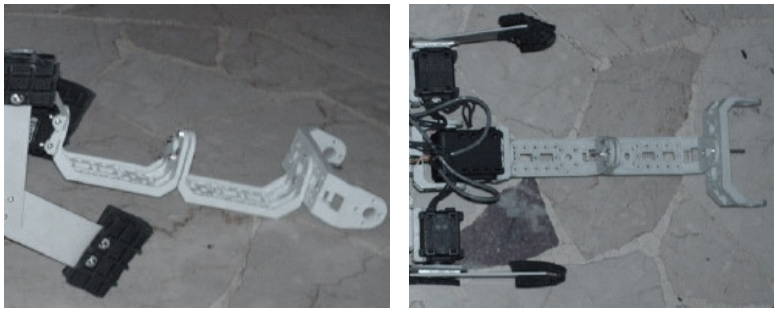
In Figura 4.9a è possibile osservare il geco mentre affronta una superficie scivolosa molto larga, si piega all'indietro e grazie al contatto della coda con la parete, evidenziata dalla linea gialla, riesce a riprendere l'equilibrio e a non cadere. In Figura 4.9b si può osservare il comportamento del robot RiSE, un robot quadrupede, mentre usa la coda attivamente per prevenirne la caduta e per assisterlo durante la scalata come se fosse un arto di emergenza. I rettangoli rossi evidenziano il comportamento del geco e di RiSE durante la fase in cui il corpo si piega all'indietro di $\approx 60°$.

4.3.2 La coda di LionHell II

LionHell II è un robot mobile che deve affrontare ostacoli di piccole o medie dimensioni, e talvolta si ritrova costretto a dover superare anche ostacoli di grandi dimensioni. Affinché tale operazione abbia successo, è necessario che il robot non cada all'indietro a causa dei ripetuti contraccolpi causati dal movimento dei Wheg che cercano un appiglio su cui poter fare forza e sollevare il resto del corpo del robot. L'aggiunta della coda garantisce una maggiore stabilità durante le salite e gli permette di sollevare il corpo

Figura 4.10: LionHell: coda originale

(a) Laterale (b) Alto



prevenendone la possibilità di cadere all'indietro, come nell'esempio del geco precedentemente mostrato. Le caratteristiche morfologiche che permettono al robot di muoversi con maggiore facilità, affrontando anche grossi ostacoli, sono:

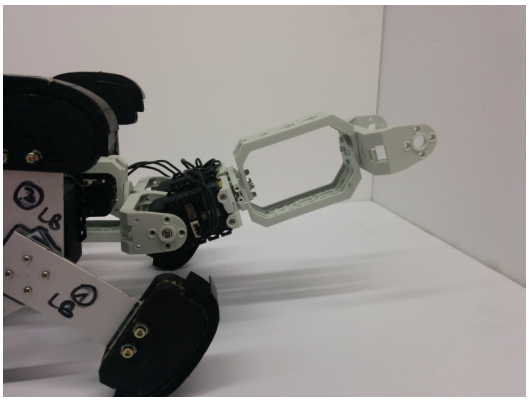
- due motori in modalità di posizione, che permettono alla coda di piegarsi e fare forza sulla punta quando è richiesto il suo intervento;
- la lunghezza totale della coda, proporzionale alla lunghezza del corpo;
- la risposta della coda che interviene solo quando il robot si sta apprestando ad affrontare grossi ostacoli, riconoscibili tramite i sensori della testa.

Rispetto al progetto originale, la coda ha subito delle modifiche volte a potenziare il robot ed aumentare l'efficacia e la forza con cui la coda preme sul terreno, ossia:

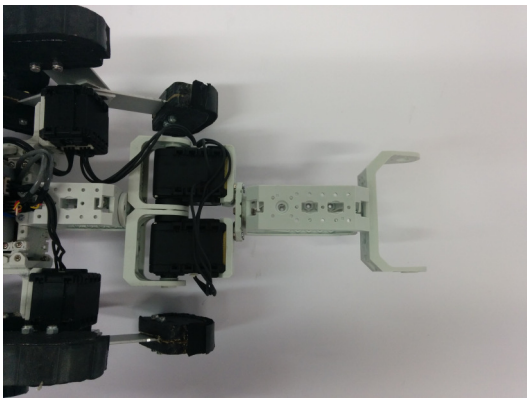
- la struttura della coda è stata rinforzata, in modo da evitare che la nuova coda si danneggi nel tempo a causa della forza sviluppata dai motori e del peso stesso del robot;
- la forza del giunto che permette alla nuova coda di muoversi è stata incrementata, utilizzando due motori in parallelo che permettono a LionHell II di muoversi con maggiore facilità mentre affronta grossi ostacoli.

Figura 4.11: LionHell II: coda rinforzata

(a) Laterale



(b) Alto



Capitolo 5

Telecontrollo in LionHell II

LionHell II è un robot mobile con straordinarie capacità di scalare, è in grado di sollevare la testa quando si trova di fronte ad un ostacolo per poter analizzare l'altezza dell'oggetto che si appresta a scavalcare in modo tale da decidere l'approccio migliore, scegliere la velocità dei motori e la forza da imprimere nella coda nel caso in cui questa sia richiesta.

Affinché LionHell II sia però in grado di esplorare efficacemente il mondo circostante, è stato necessario utilizzare un controllo remoto tramite l'utilizzo di un particolare componente chiamato XBee. Tale decisione è stata presa a causa delle ridotte capacità sensoriali della testa, che nonostante sia in grado di riconoscere la presenza di ostacoli e di scegliere la strategia migliore per affrontarli, rimane comunque insufficiente per esplorare l'ambiente circostante scegliendo la strada migliore, come mostrato nel Capitolo 3, Sezione 3.2.

La Sottosezione 5.1 descrive il telecomando utilizzato, il funzionamento e la componentistica.

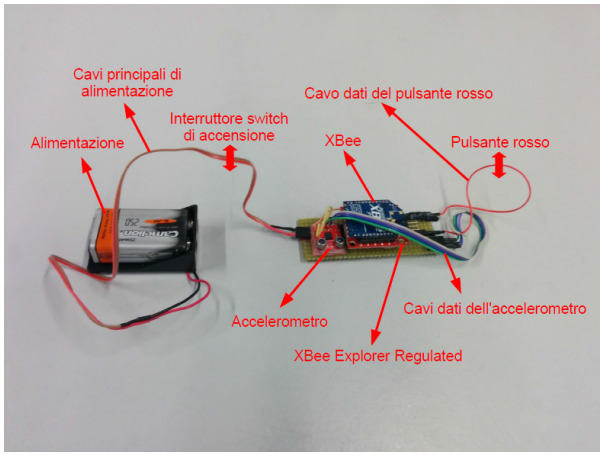
La Sottosezione 5.2 descrive l'XBee installato su LionHell II e le modifiche che sono state apportate affinché funzionasse correttamente.

La Sottosezione 5.3 descrive le modifiche che sono state effettuate nel codice per ottenere il movimento desiderato.

Figura 5.1: Telecomando

(a) Esterno

(b) Interno



5.1 XBee del telecomando

La Figura 5.1 mostra la componentistica basilare del telecomando (escluso il bottone e un interruttore switch, che per motivi pratici non sono mostrati in Figura in quanto sono incorporati nella struttura esterna del telecomando). Il telecomando è costituito da:

- una batteria da 9 V e 250 mAh che costituisce l'alimentazione;
- un XBee Explorer Regulated che si occupa della regolazione della tensione a 3.3V, del condizionamento del segnale e degli indicatori di attività di base e converte i segnali da 5V a 3.3V in modo da poter collegare al sistema un qualsiasi modulo XBee;
- un XBee 4214A che svolge il ruolo di trasmettere i segnali ricevuti in ingresso e comunicarli all'XBee installato sul corpo di LionHell II;
- un accelerometro analogico a tre assi ADXL335, con scheda di rilevamento +/-3g privo di regolatore di tensione (la tensione in ingresso deve essere compresa tra 1.8Vdc e 3.6Vdc);
- un interruttore switch che svolge il ruolo di interruttore di accensione (ON) e spegnimento (OFF);

- un pulsante rosso instabile normalmente aperto, che permette l'utilizzo del telecomando solo mentre è premuto.

Il movimento di LionHell II avviene tramite l'inclinazione del telecomando (puntando la punta del cappello da mago in avanti e con il pulsante rosso verso l'alto il robot sta fermo, mentre alzando o abbassando il cappello è possibile farlo andare rispettivamente in avanti o indietro, ed inclinando il telecomando a destra o sinistra il robot ruota a destra o a sinistra rispettivamente), il quale rileva una variazione degli assi X e Y per mezzo dell'accelerometro. I valori così letti sono poi passati all'XBee installato sul telecomando, il quale li invia direttamente all'XBee su LionHell II.

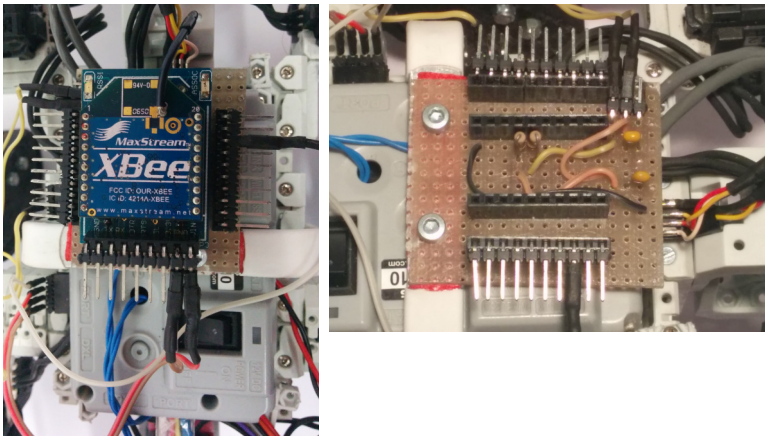
5.2 XBee di LionHell II

L'obiettivo dell'XBee è ricevere i dati dal telecomando e inviare i dati così ricevuti alla scheda di LionHell II, con il risultato che la scheda non si accorge neanche dell'esistenza dell'XBee, come se leggesse direttamente i dati dal telecomando.

La Figura 5.2a mostra l'XBee 4214A utilizzato in LionHell II, montato direttamente sopra la scheda di controllo CM-510, al centro del corpo del robot, mentre la Figura 5.2b mostra i componenti sottostanti l'XBee. Come si può osservare, sono presenti (per ciascuno dei due ingressi digitali) una resistenza e un condensatore: il motivo è presto spiegato. I dati trasmessi dall'XBee del telecomando all'XBee di LionHell II sono in forma analogica, ma gli ingressi della scheda di controllo richiedono un'entrata digitale non di tipo onda quadra (ottenibile tramite una porta invertente NOT, un trigger di Schmitt, un condensatore e un resistore) ma tramite un filtro di tipo passa-basso (che richiede solamente l'utilizzo di un circuito RC, basato appunto sull'utilizzo di un resistore e un elemento dinamico, il condensatore).

Figura 5.2: XBee su LionHell II

(a) XBee (b) Logica sottostante



5.3 Modifiche al Firmware

Una volta che il segnale è partito dal telecomando, è passato nell'XBee installato su LionHell II ed è stato opportunamente modificato in modo da restituire i valori originali letti inizialmente dall'accelerometro del telecomando, è la volta della scheda di controllo CM-510.

La scheda si comporta come se leggesse i valori direttamente dall'accelerometro, senza neppure accorgersi dell'esistenza di tutti i componenti intermedi, e in base a questi valori discrimina le azioni da compiere. Di seguito è riportato il codice di LionHell II riguardante la lettura dei valori dell'accelerometro:

```
// Read Remote Controller via XBee using Virtual Wires
{

        resultX = adc_start ( 4 );
        resultY = adc_start ( 3 );
        bRemoteButton = (PINE & BTN_RIGHT);
        //printf("\r\nresultX resultY button : %u %u %d",
            resultX, resultY, bRemoteButton);

        // BUTTON
        if (bRemoteButton)
        {
```

```
                walking = true;
        }
        else
        {
                walking = false;
        }

        //X
        if ( resultX > 340 )
        {
                turnL = 0; turnR=0; // Go Fwd
        } else
        if ( resultX < 300  )
        {
                turnL = 1; turnR=1; // Go Bwd
        }

        //Y
        if ( resultY > 340 )
        {
                turnR = 1; turnL=0; // Turn Right
        } else
        if ( resultY < 300 )
        {
                turnL = 1; turnR=0; // Turn Left
        }
}
```

Il codice mostra una prima lettura dei valori di X e Y dell'accelerometro, salvati rispettivamente nelle variabili `resultX` e `resultY`, successivamente viene letto il valore del pulsante rosso tramite la variabile `bRemoteButton`. Nel caso in cui il pulsante rosso sia premuto allora la variabile `walking` (cammina) viene settata a `true` (vero) ed in base ai valori di `resultX` e `resultY` viene scelta la direzione da prendere.

Il codice riportato di seguito mostra invece il comportamento di LionHell II dopo la ricezione dei segnali dell'accelerometro e dopo che è stata scelta l'azione da compiere:

```
//Walking Actions
```

```
        if (walking==1){
                if(turnL && turnR){
                int i ;
                for (i=0;i<3;i++){//Go Backward
                        dxl_write_word( whegs_sx[i],
                            P_MOVING_SPEED_L, 1624 );
                        dxl_write_word( whegs_dx[i],
                            P_MOVING_SPEED_L, 600 );
                }
        }else
        if(turnL){//Turn Left
                int i ;
                for (i=0;i<3;i++){
                        dxl_write_word( whegs_sx[i],
                            P_MOVING_SPEED_L, 1624 );
                        dxl_write_word( whegs_dx[i],
                            P_MOVING_SPEED_L, 1624 );
                        //dxl_write_word( whegs_dx[i],
                            P_MOVING_SPEED_L, 0 );
                }
        }else
        if(turnR){//Turn Right
                int i ;
                for (i=0;i<3;i++){
                        dxl_write_word( whegs_sx[i],
                            P_MOVING_SPEED_L, 600 );
                        //dxl_write_word( whegs_sx[i],
                            P_MOVING_SPEED_L, 0 );
                        dxl_write_word( whegs_dx[i],
                            P_MOVING_SPEED_L, 600 );
                }
        }else
        if (!(turnL + turnR)){//If not turning
                go_fwd();//Restart walking
        }
}else{//Stop Walking
        stop();
}
```

In questo caso la scelta di muovere LionHell II si basa esclusivamente sui

valori di `walking` (cammina), `turnL` (gira a sinistra) e `turnR` (gira a destra). Nel caso in cui i valori di `turnL` e `turnR` siano entrambi a zero, allora LionHell II proseguirà dritto (la funzione `dxl_write_word` imposta la velocità di movimento di ogni singolo Wheg e la sua direzione). Negli altri due casi, invece, a differenza dei valori di `turnL` e `turnR`, il robot prenderà la decisione di girare a sinistra o a destra modificando opportunamente le velocità e le direzioni dei Wheg.

Capitolo 6

Design di LionHell II

LionHell II è un robot di esplorazione, ma anche l'aspetto visivo del robot svolge un ruolo importante, in quanto un robot con aspetto umanoide o animale è in genere accettato più facilmente confronto ad un robot privo di tali caratteristiche. In questa sezione ci occuperemo dunque di descrivere le modifiche che sono state effettuate a livello di design, motivando la ragione delle nostre scelte.

La Sezione 6.1 descrive il ruolo del design nella costruzione di un robot, e come il robot stesso viene percepito in maniera differente a differenza del suo aspetto.

La Sezione 6.2 descrive e mostra il volto di LionHell II.

La Sezione 6.3 descrive l'aspetto esteriore del telecomando.

La Sezione 6.4 descrive la nuova corazza che copre il busto e protegge le componenti sottostanti e mostra la stessa corazza nel mondo animale e nelle armature del passato.

La Sezione 6.5 mostra il pulsante che è stato aggiunto in seguito all'aggiunta della corazza precedentemente descritta.

6.1 Perché il design è importante

Gli esseri umani tendono ad esibire un naturale interesse ed attrazione verso le altre specie, e tale interesse è chiamato Biophilia Hypothesis [32, 41, 22, 46], che significa letteralmente "amore per la vita o per i sistemi viventi". Il primo ad usare tale termine fu Erich Fromm e fu utilizzato per descrivere l'orientamento fisiologico che ci porta ad essere attratti da tutto ciò che è vivo e vitale.

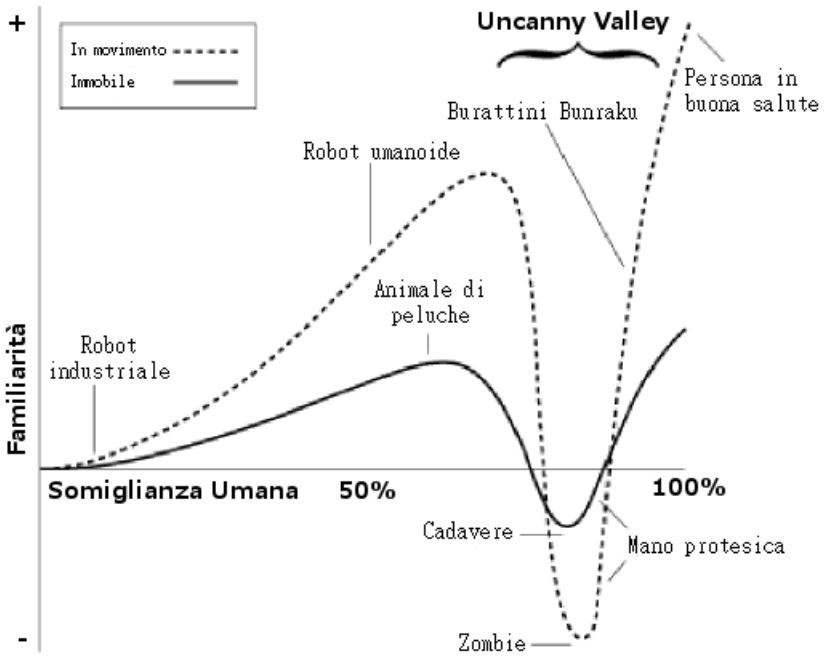
Nonostante le numerose e recenti ricerche riguardanti le iterazioni uomo-animale [22] (come i benefici derivanti dal contatto con gli animali durante l'età dello sviluppo), è stata posta poca attenzione all'identificazione delle specifiche caratteristiche animali che provocano risposte differenti negli umani, in particolare nei bambini. Gli studi preliminari hanno analizzato in particolare l'iterazione dei bambini nei confronti della robotica e nei confronti di animali, considerando la loro risposta emozionale e la volontà di impegnarsi in una relazione sociale.

Inoltre, le nuove ricerche provenienti dall'università Georgia Institute of Technology, seguite dallo studente graduato di psicologia Akanksha Prakash [41, 19, 62], mostrano la volontà da parte degli adulti anziani di accettare robot nelle loro vite quotidiane e le loro scelte basate sull'aspetto che il robot dovrebbe avere dipendono dalle funzioni che il robot stesso deve svolgere (le persone più anziane e più giovani hanno infatti dimostrato maggiore interesse nei confronti di un aspetto esteriore prevalentemente robotico-meccanico nel caso in cui il ruolo del robot dovesse rientrare nelle attività di cura personale come fare il bagno, una preferenza basata esclusivamente su problemi di privacy). Inoltre, se il robot non dovesse esibire un comportamento socialmente accettabile, le persone potrebbero non accettarlo, considerandolo noioso, spaventoso. Di conseguenza:

> Come può un robot comportarsi un maniera socialmente accettabile?

Generalmente, i robot che presentano un aspetto prevalentemente meccanico tendono ad essere trattati con meno rispetto e in maniera meno socialmente

Figura 6.1: Il diagramma di Mori



interattiva al confronto di quei robot aventi aspetto umanoide o animale, ciononostante ad un certo punto l'effetto diventa repulsivo a causa del fatto che il robot risulta avere troppe caratteristiche umane (od animali), riprodotte però in maniera grottesca. Tale effetto è illustrato nel diagramma di Mori [64, 54], in Figura 6.1, dove la curva che precipita sotto l'asse genera un'area definita con il nome di "Uncanny Valley", ossia valle sconcertante, per indicare l'effetto repulsivo generato da questa.

Inoltre, l'aspetto del robot provoca una certa aspettativa da parte dell'osservatore. Se l'aspetto del robot è troppo avanzato per le reali capacità espressive del robot, le persone tenderanno a giudicarlo disonesto, in quanto vi è una discordanza percepibile tra i segnali emessi dal robot e quelli inconsciamente valutati come corretti dagli umani. Il robot, dunque, dovrebbe avere un comportamento sociale e un aspetto in accordo con le funzionalità e le capacità del robot stesso.

Infine, in Figura 6.2 è possibile osservare tre robot [64, 54] aventi caratteristiche più o meno umanoidi:

Figura 6.2: I tre robot usati per nell'esperimento

(a) Meccanoide (b) Umanoide (c) Androide

Meccanoide in Figura 6.2a si può osservare un robot avente un aspetto prevalentemente meccanico. Il robot non presenta alcuna caratteristica umanoide, e comunica tramite segnali acustici;

Umanoide in Figura 6.2b vi è invece un robot con un aspetto umanoide non realistico, ed è percepito come robot da tutte le persone che vi hanno interagito. Il robot è dotato di alcune caratteristiche umane semplificate, come una testa, occhi, sopracciglia, braccia e gambe, comunica per mezzo di una voce sintetizzata di bassa qualità;

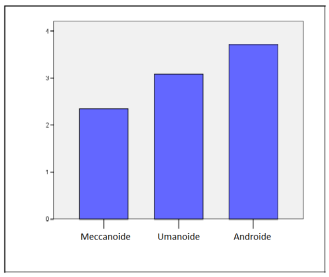
Androide In Figura 6.2c il robot esibisce un aspetto ed un comportamento che sono simili a quelli effettivamente mostrati da un essere umano, comunica tramite una voce sintetizzata di alta qualità.

L'obiettivo dei tre robot è lo stesso: una volta che una persona citofona in casa, il robot raggiunge l'inquilino della casa e gli comunica che vi è qualcuno alla porta. I test che sono stati effettuati (Figura 6.3) hanno dimostrato che il robot androide è stato percepito come migliore confronto agli atri due robot su tutti i punti di vista, considerando aspetti quali: estroversione, stabilità emotiva, coscienziosità, gradevolezza e intelligenza, nonostante l'unica cosa che cambiasse fosse solamente l'aspetto esteriore e non la logica di comportamento.

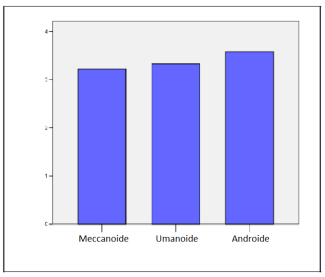
Nel caso di LionHell II, si è optato ad un aspetto prevalentemente animale, data la forma del corpo del robot, la presenza di sei Wheg e di una coda.

Figura 6.3: Punteggi medi dei tre robot percepiti

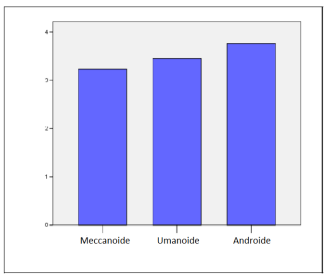
(a) Estroversione



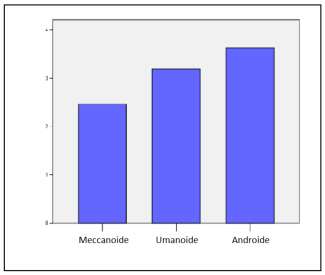
(b) Stabilità emotiva



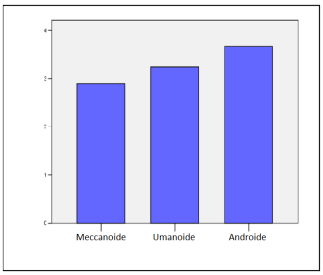
(c) Coscienziosità



(d) Gradevolezza



(e) Intelligenza



Le modifiche sostanziali sono state apportate prevalentemente alla testa e al busto di LionHell II, come mostrato nelle sottosezioni successive, in cui ne è descritto in dettaglio ogni singolo aspetto.

6.2 Volto di LionHell II

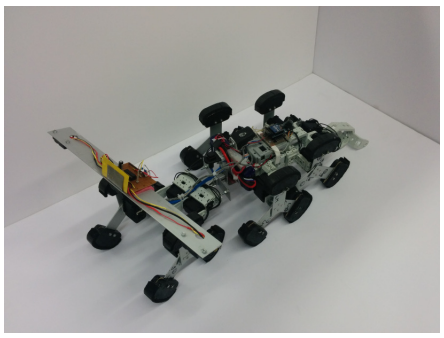
LionHell significa letteralmente “leone inferno”, traducibile anche come Leone Infernale. Di conseguenza è stato d’obbligo cercare di dargli un aspetto felino, nonostante la barra sensoriale molto lunga che costituisce la testa del robot. Per questa ragione è stata scelta come esempio la tigre reale del Bengala (Panthera Tigris Tigris [6]), la tigre più diffusa e più comune, in quanto si sono avuti problemi pratici nel cercare di creare una vera e propria cresta intorno alla testa.

Il nuovo volto è costituito da un materiale plastico, molto leggero, che permette di abbellire LionHell II senza incrementargli il peso in maniera eccessiva,

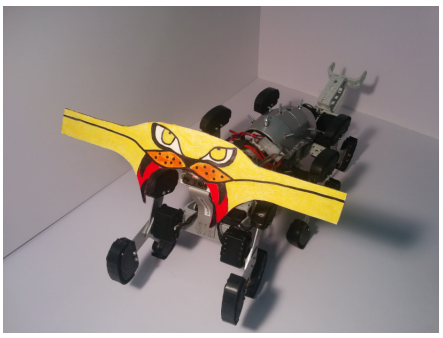
Figura 6.4: Volto di LionHell II

(a) Sagoma del volto

(b) LionHell II senza volto



(c) LionHell II con il volto



mentre i colori sono stati applicati tramite pennarelli indelebili, lasciando lo spazio sufficiente per il sensore centrale (e questo è il motivo della bocca ad arco, come si può osservare in Figura 6.4a). In Figura 6.4b è possibile osservare l'aspetto di LionHell II senza protezioni e in Figura 6.4c lo stesso LionHell II dotato di volto e corazza protettiva per il busto.

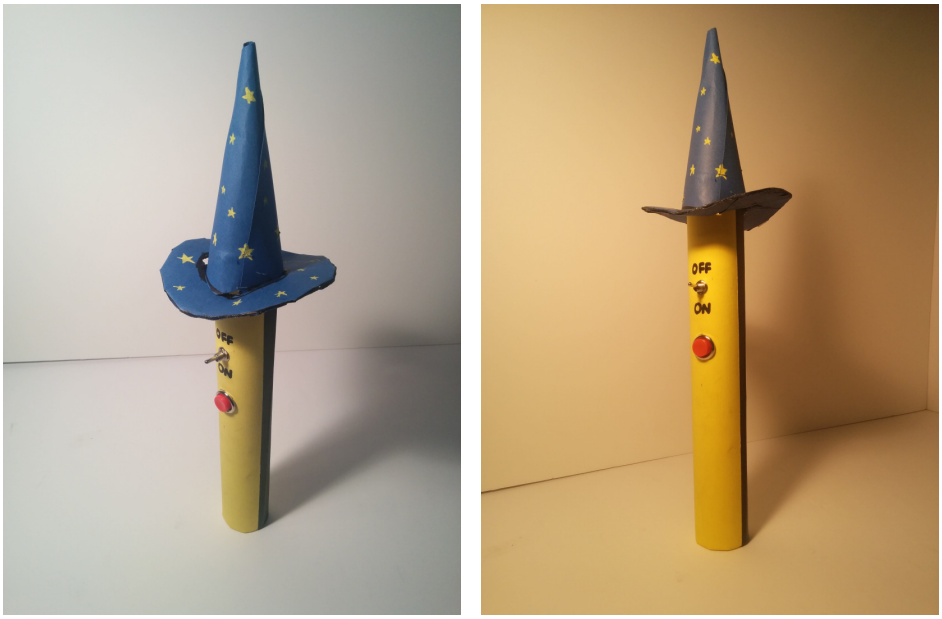
6.3 Il telecomando di LionHell II

LionHell II è controllabile tramite l'utilizzo di un telecomando (mostrato in Figura 6.5 e descritto in dettaglio nel Capitolo 5, Sezione 5.1). Per cercare di rendere il telecomando appetibile per un vasto pubblico, si è pensato di colorare il telecomando e di aggiungere un classico cappello da mago, blu con stelle gialle.

La componentistica del telecomando è racchiusa in un tubo di plastica dura, ricoperto da un cartoncino giallo (da cui fuoriescono l'interruttore e il pulsante rosso), la batteria è estraibile dalla parte posteriore del telecomando mentre rimuovendo il cappello e l'elastico sottostante, è possibile estrarre (con molta cautela) i restanti componenti, ed in particolare l'XBee programmabile. Il cappello è fatto in cartone blu, mentre le stelle sono state disegnate con un pennarello giallo indelebile e la tesa è rinforzata internamente con un sottile strato in alluminio.

Figura 6.5: Il telecomando-bacchetta

(a) Dall'alto (b) Lateralmente



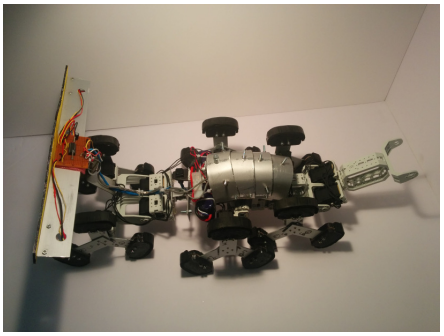
6.4 La corazza di LionHell II

LionHell II è stato dotato di una struttura metallica, una corazza in alluminio, per proteggere il corpo, ed in particolare la scheda di controllo CM-510 e il delicato XBee da possibili involontarie cadute, garantendogli una protezione solida ed efficacie. Come si può osservare in Figura 6.6, la corazza è composta da 4 placche semicircolari sovrapposte, bloccate le une sulle le altre da alcune viti posizionate in basso, mentre l'interno è stato ricoperto di gomma in modo da evitare un possibile cortocircuito tra i pin della scheda dell'XBee che potrebbero venire accidentalmente in contatto con la corazza.

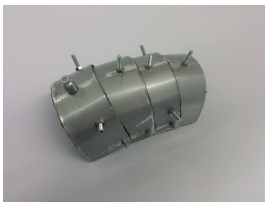
In figura si può inoltre osservare la presenza di una vite di dimensioni più grandi, in corrispondenza della scaglia più larga: si tratta della vite di rimozione della corazza (di cui è disponibile anche una manovella nel caso in cui la vite sia stata stretta con troppa forza), necessaria per poter interagire con la scheda di controllo CM-510 con il cavo di programmazione e per poter rimuovere e riprogrammare l'XBee.

Figura 6.6: Corazza di LionHell II

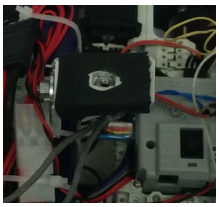
(a) LionHell II con la corazza



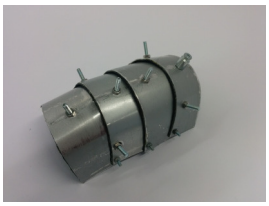
(b) Corazza di fronte



(c) Innesto



(d) Corazza di spalle

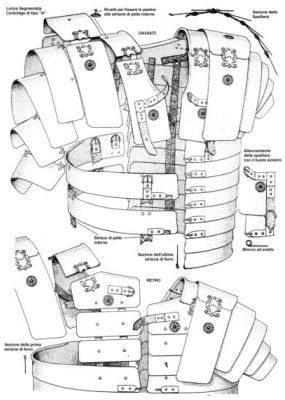


La forma della corazza è stata ottenuta imitando le corazze degli animali e le armature del passato, come mostrato in Figura 6.7. La Figura 6.7a mostra un'armatura lorica segmentata [4], un tipo di armatura utilizzata in periodo romano, che mostra chiaramente la disposizione delle varie placche metalliche, tra loro sovrapposte. La Figura 6.7b mostra un onisco [1] (Armadillidium Vulgare, conosciuto anche con il nome di porcellino di terra o porcellino di Sant'Antonio), un crostaceo capace di colonizzare la terra: l'onisco usa la sua corazza per proteggersi se si sente in pericolo o minacciato, appallottolandosi e formando una sfera che non offre appigli all'aggressore. La Figura 6.7c mostra invece lo scheletro di un armadillo [21] (Euphractus Sexcinctus, conosciuto anche come Armadillo a sei fasce) e la sua corazza, costituita da spesse placche ossee che possono essere utilizzate come protezione, come nel caso dell'onisco precedentemente descritto.

L'aggiunta dei chiodi che fuoriescono di alcuni centimetri dalla corazza servono inoltre a dare l'effetto dell'armatura corazzata e aggiungono spessore alla corazza e all'intero robot, migliorando l'effetto visivo generale.

Figura 6.7: Corazze varie

(a) Lorica segmentata



(b) Onisco

(c) Armadillo

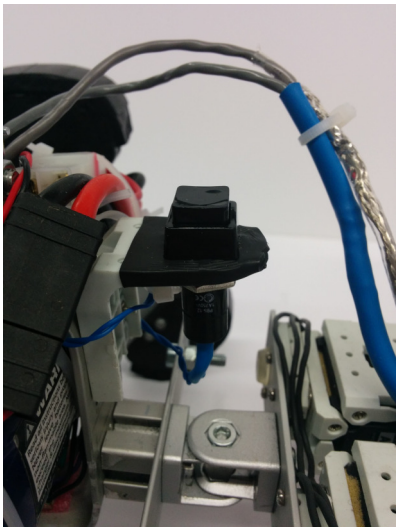
6.5 Il pulsante di accensione

L'aggiunta della corazza centrale ha comportato l'impossibilità di accedere al pulsante di accensione a meno di rimuovere e rimontare la corazza ogni volta. La soluzione è stata quella di montare un pulsante stabile normalmente aperto, come mostrato in Figura 6.8a all'inizio della corazza (Figura 6.8b), in corrispondenza del centro del robot.

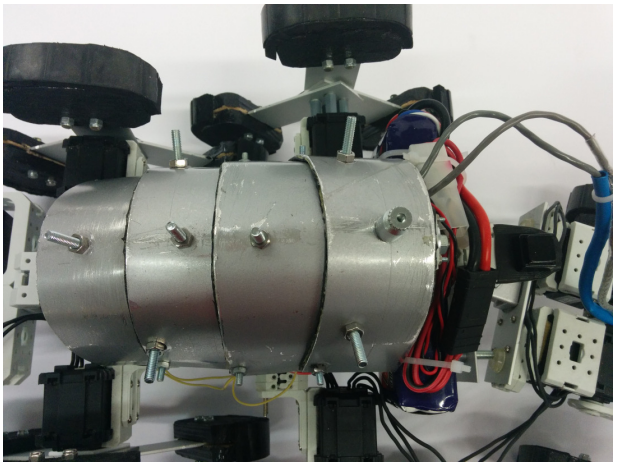
Il nuovo pulsante è direttamente connesso alla scheda di controllo CM-510 per mezzo del cavo blu che si vede in figura, permettendone un accesso facile ed intuitivo.

Figura 6.8: Pulsante di accensione

(a) Particolare



(b) Corazza e pulsante



Capitolo 7

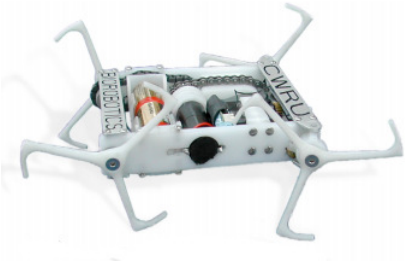
Conclusioni

LionHell II è un robot esapode che sfruttando la combinazione di Wheg e coda e di un giunto centrale è in grado di superare ostacoli di medie e grandi dimensioni. L'utilizzo dei Wheg ne ha semplificato la struttura [65], evitando di utilizzare numerosi attuatori per muovere le gambe che avrebbe dovuto avere, e il loro utilizzo combinato alla coda che è in grado di fare da supporto durante le scalate gli permette di muoversi ovunque con facilità. L'aggiunta del controllo remoto permette infine di decidere la direzione di LionHell II, scegliendo cosa è in grado di affrontare e cosa no.

Le modifiche che sono state apportate ai Wheg hanno permesso di ottenere un nuovo modello più resistente, dotato di una gomma ammortizzante per ridurre le sollecitazioni dei Wheg su terreno accidentato, durante le scalate e le discese, hanno aumentato l'area di contatto con il terreno riducendo possibili problemi derivanti dal muoversi su sabbia o su terreni soffici e la gomma garantisce una buona aderenza sul qualunque superficie. Il nuovo giunto centrale permette a LionHell II di muoversi e ruotare più velocemente, mentre la nuova coda rinforzata facilità il robot a superare ostacoli di altezza media e alta e impedisce a LionHell II di ribaltarsi. Il telecontrollo che è stato aggiunto garantisce il controllo remoto di LionHell II per mezzo di un telecomando controllato tramite accelerometro, che comunica con il robot per mezzo di XBee.

Figura 7.1: Mini-Whegs IV e Lunar Whegs

(a) Mini-Whegs IV



(b) Lunar Whegs

LionHell II presenta molti più gradi di libertà confronto ad altri robot di esplorazione in grado di muoversi su terreni molto accidentati e ciò gli garantisce un movimento più fluido, e l'utilizzo di giunto centrale motorizzato [30] permette a LionHell II di alzare la barra sensoriale per individuare la presenza o meno di ostacoli e lo aiuta nelle scalate. In Figura 7.1 è possibile osservare i robot Mini-Whegs IV [55] e Lunar Whegs [33, 5, 34]. Confronto al robot Mini-Whegs IV (Figura 7.1a), LionHell II è dotato di un maggior numero di Wheg, di un giunto centrale motorizzato e di una coda, che evita possibili ribaltamenti, svolgendo il ruolo di gamba di emergenza. In Figura 7.1b si può osservare, invece, il robot Lunar Whegs, il quale è dotato di Wheg simili a quelli installati su LionHell II e di un giunto centrale che gli permette di sollevare i Wheg anteriori, come LionHell II, ma a differenza di quest'ultimo è privo della coda di supporto e del giunto passivo.

In Figura 7.2 si può osservare LionHell II mentre curva intorno al proprio asse in senso orario, mentre in Figura 7.3 si può osservare lo stesso LionHell II mentre ritorna nella posizione iniziale ruotando intorno al proprio asse in senso antiorario. I test mostrano la buona manovrabilità che il robot possiede, riuscendo a curvare di $\sim$180° intorno al proprio asse in meno di dieci secondi su piastrella. In Figura 7.4, in Figura 7.5 e in Figura 7.6, LionHell II affronta rispettivamente un ostacolo basso (6 cm), medio (10 cm) e alto (16 cm), e durante la fase di scalata dell'ostacolo è possibile osservare l'utilizzo della coda come punto di appoggio. In Figura 7.7 LionHell II affronta una semplice rampa (altezza massima 12 cm), mentre in Figura 7.8 LionHell II

affronta la stessa rampa dalla direzione opposta. In Figura 7.9 LionHell II si appresta ad affrontare un tubo di dimensioni medie (diametro 10 cm), riuscendo ad attraversarlo, mentre in Figura 7.10 supera con successo una serie di oggetti sparsi casualmente sul terreno (si possono notare sbarre di metallo, pezzi di metallo dalla forma ad U e ad L, pezzi di legno e gommapiuma, talvolta sovrapposti).

La struttura corrente non consente, comunque, la creazione di un robot che sia in grado di muoversi ed esplorare l'ambiente autonomamente, in quanto necessiterebbe di numerosi sensori a infrarossi, simili a quelli usati nella barra sensoriale, tutti intorno al robot, in modo che LionHell II sia in grado di rilevare la presenza di ostacoli intorno a sé. La barra sensoriale frontale è insufficiente per esplorare l'ambiente, come descritto nel Capitolo 3, Sezione 3.2, e l'aggiunta di telecamere sulla barra sensoriale stessa ne aumenterebbe eccessivamente il peso. Di conseguenza, nel caso si vogliano aggiungere delle telecamere, per fare in modo che LionHell II sia in grado di muoversi autonomamente o per usarle per un eventuale controllo remoto da computer, è necessario cercare un punto del robot in cui sia possibile installarle senza che queste si possano danneggiare nel caso in cui il robot cada lateralmente o di schiena a causa di un movimento improvviso. I motori AX-12, inoltre, sono inadatti per il moto continuo dei Wheg, a causa della banda morta che il motore stesso presenta e in quanto non sono in grado di sviluppare una forza elevata a velocità basse, inoltre la struttura degli attuatori è in plastica di ingegneria, e il movimento stesso dei Wheg tende, nel tempo, a danneggiarne la meccanica interna.

In conclusione, LionHell II è un robot dotato di una struttura singolare che gli permette di muoversi adeguatamente sfruttando la componentistica di cui è provvisto, ma necessita di diversi cambiamenti nel caso in cui si volesse creare un robot totalmente autonomo e non controllato in remoto come in questo caso.

LionHell II ha dimostrato di possedere una buona manovrabilità e la capacità di superare numerosi ostacoli, sfruttando una coda che svolge il ruolo di supporto durante le scalate e del giunto centrale motorizzato che gli permette

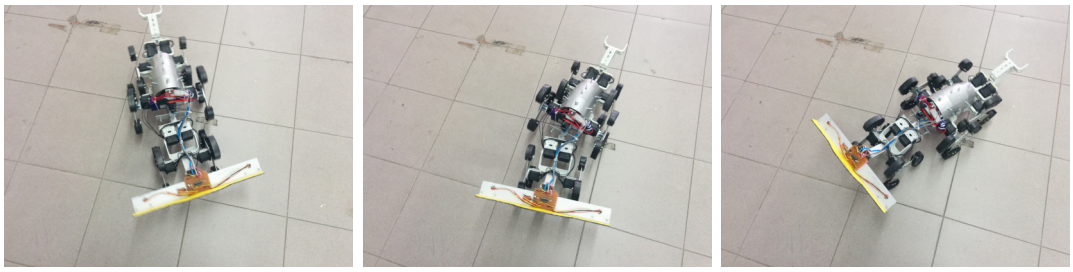
di alzare la parte frontale. Equipaggiato con una telecamera per un controllo remoto da computer, LionHell II sarebbe in grado di esplorare edifici pericolanti, superare gli ostacoli che incontra lungo il suo percorso e potrebbe dunque essere utilizzato per esplorare cunicoli e, in generale, troverebbe impiego in ogni ambito in cui è richiesto un robot con discrete capacità di scalare.

Studi recenti [61, 25] hanno inoltre dimostrato la possibilità di utilizzare robot con abilità esplorative su terreni accidentati per applicazioni swarm, ossia per la creazione di numerosi robot molto economici capaci di comportarsi come sciami ed essere quindi in grado di collaborare per raggiungere obiettivi comuni, come spostare grossi oggetti o esplorare rapidamente una vasta zona creando una mappa dell'area comune a tutti i robot.

Figura 7.2: LionHell II: rotazione in senso orario

(a) (b) (c)

(d) (e) (f)



(g)

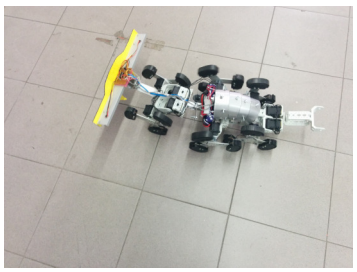


Figura 7.3: LionHell II: rotazione in senso antiorario

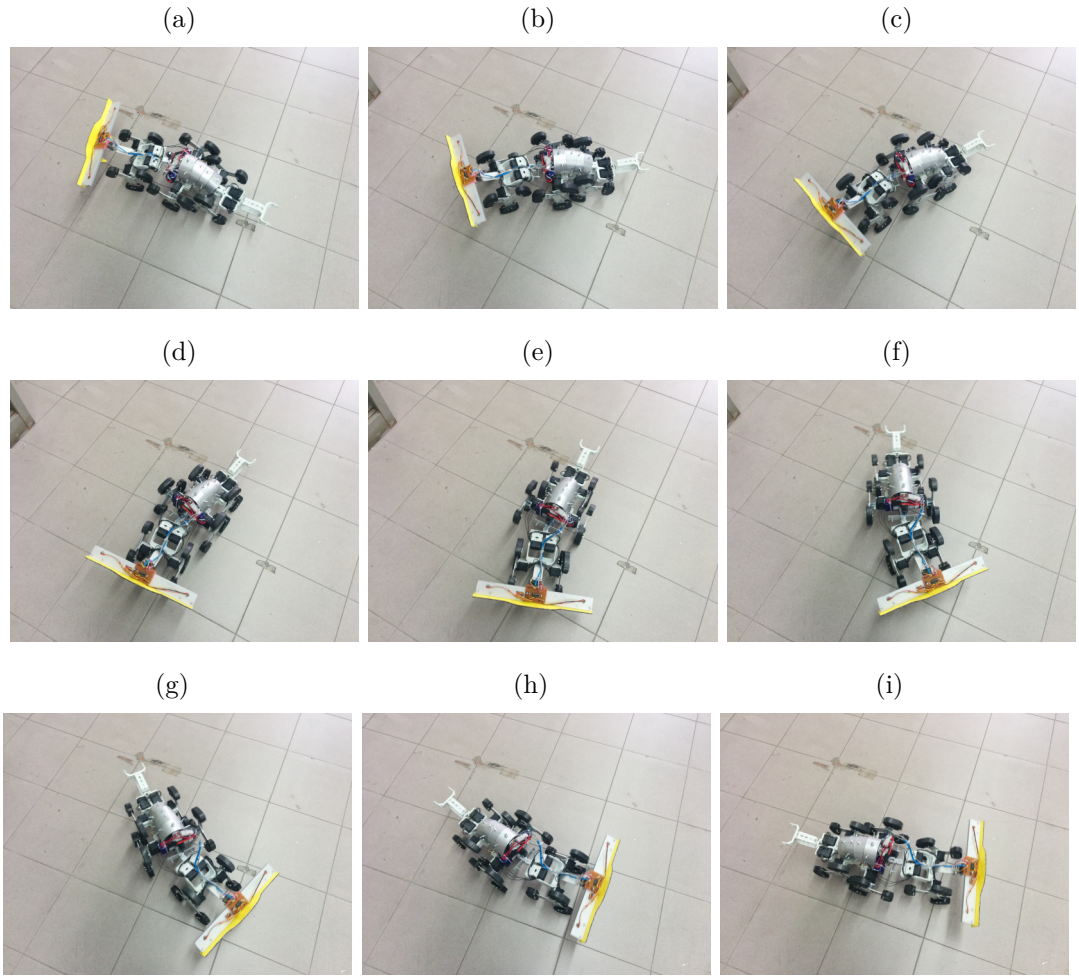
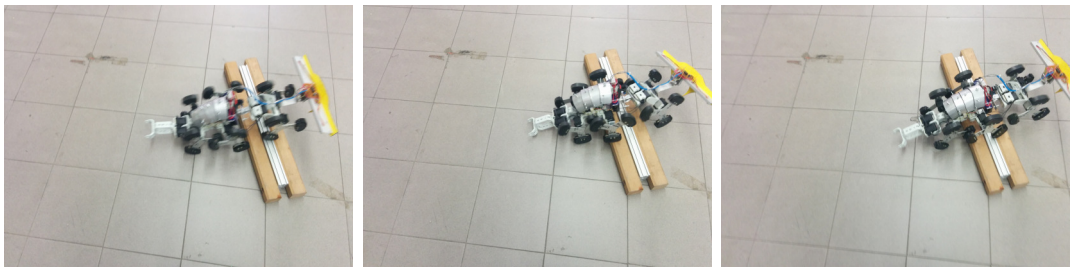


Figura 7.4: LionHell II: ostacolo basso

(a) (b) (c)

(d) (e) (f)



(g) (h) (i)

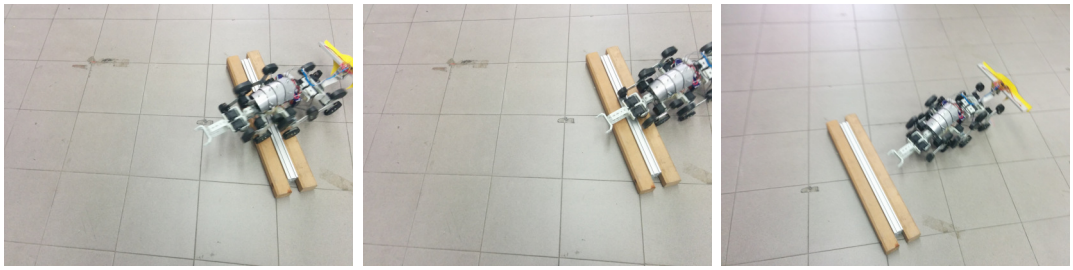


Figura 7.5: LionHell II: ostacolo medio

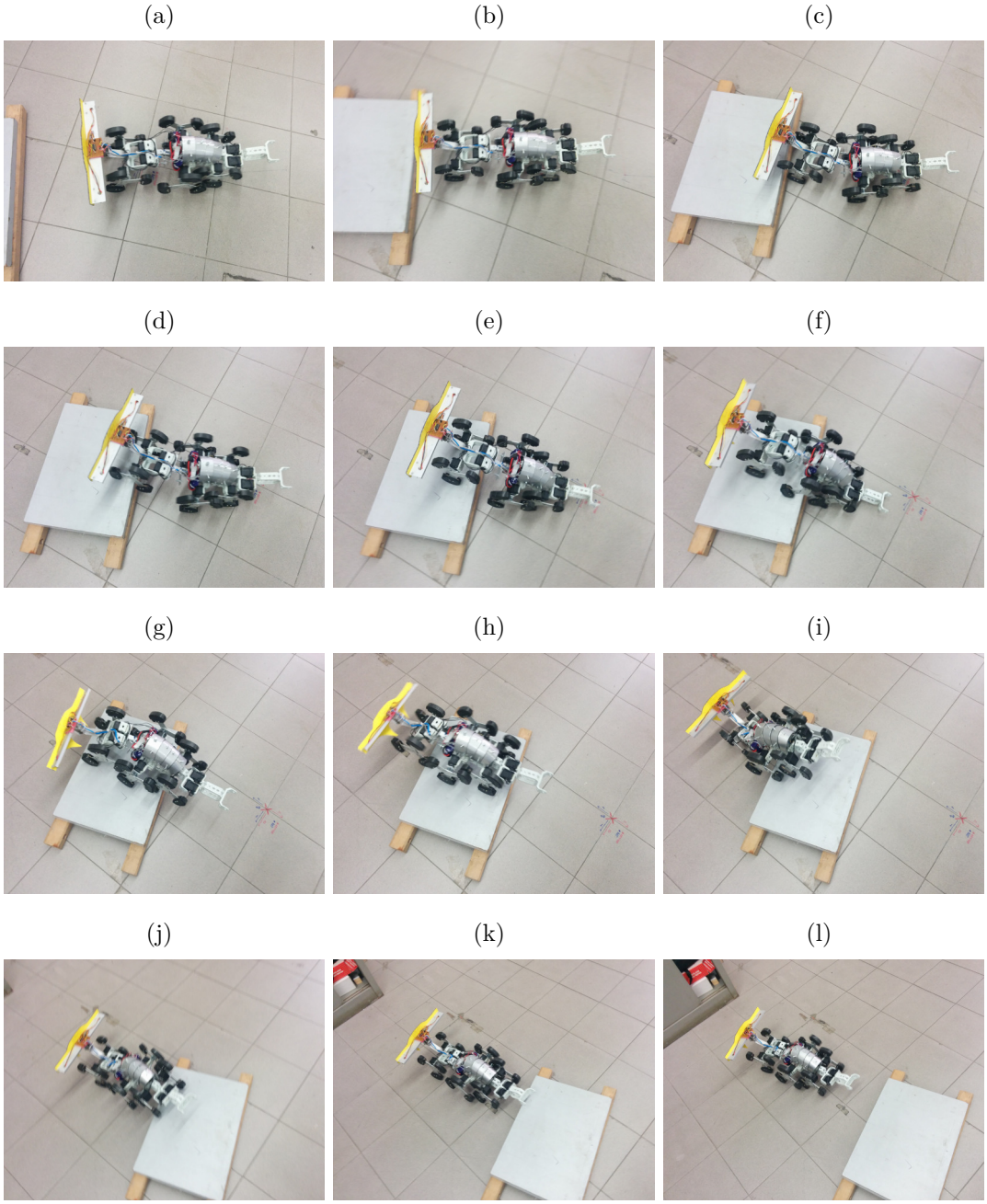


Figura 7.6: LionHell II: ostacolo alto

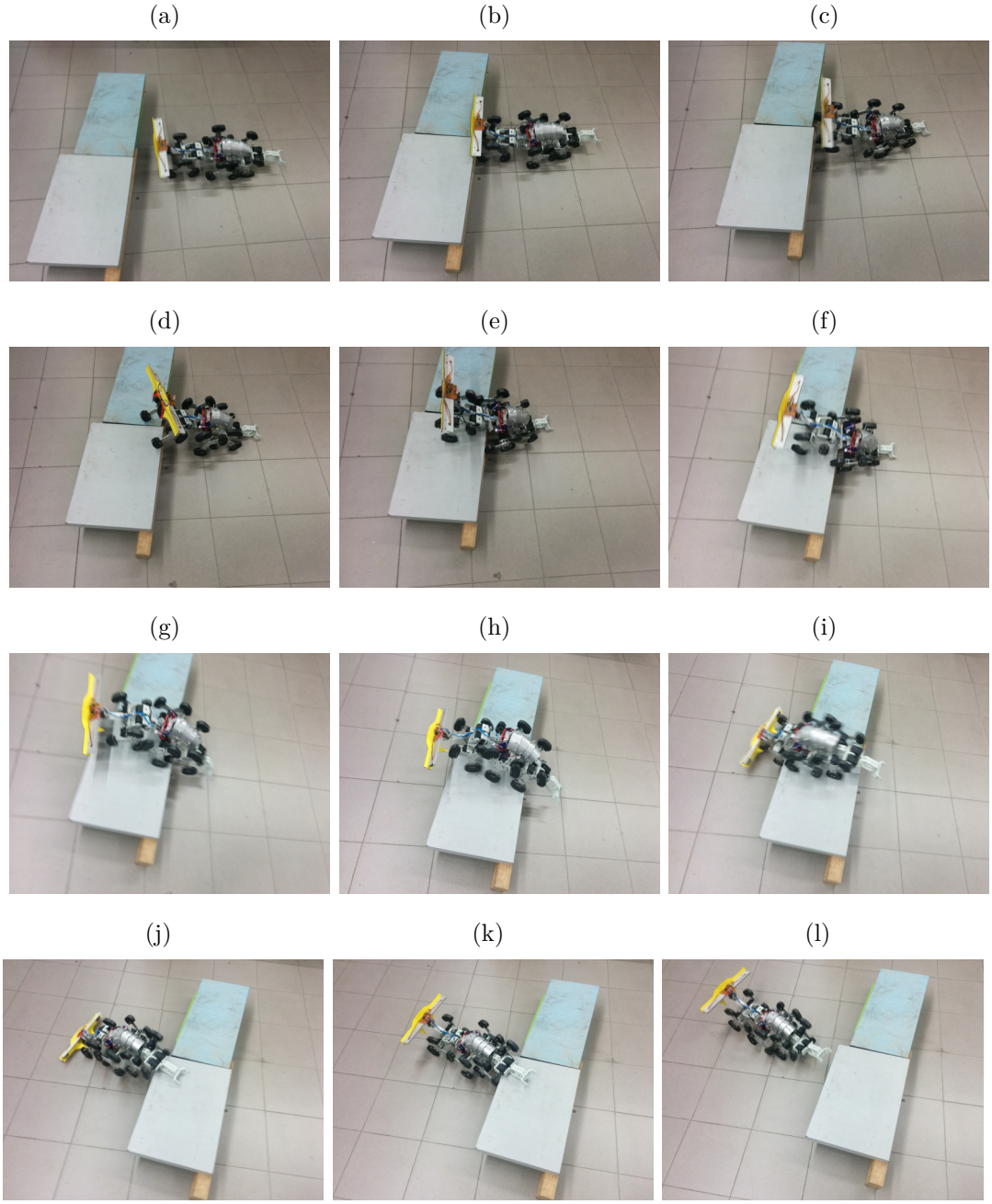


Figura 7.7: LionHell II: rampa

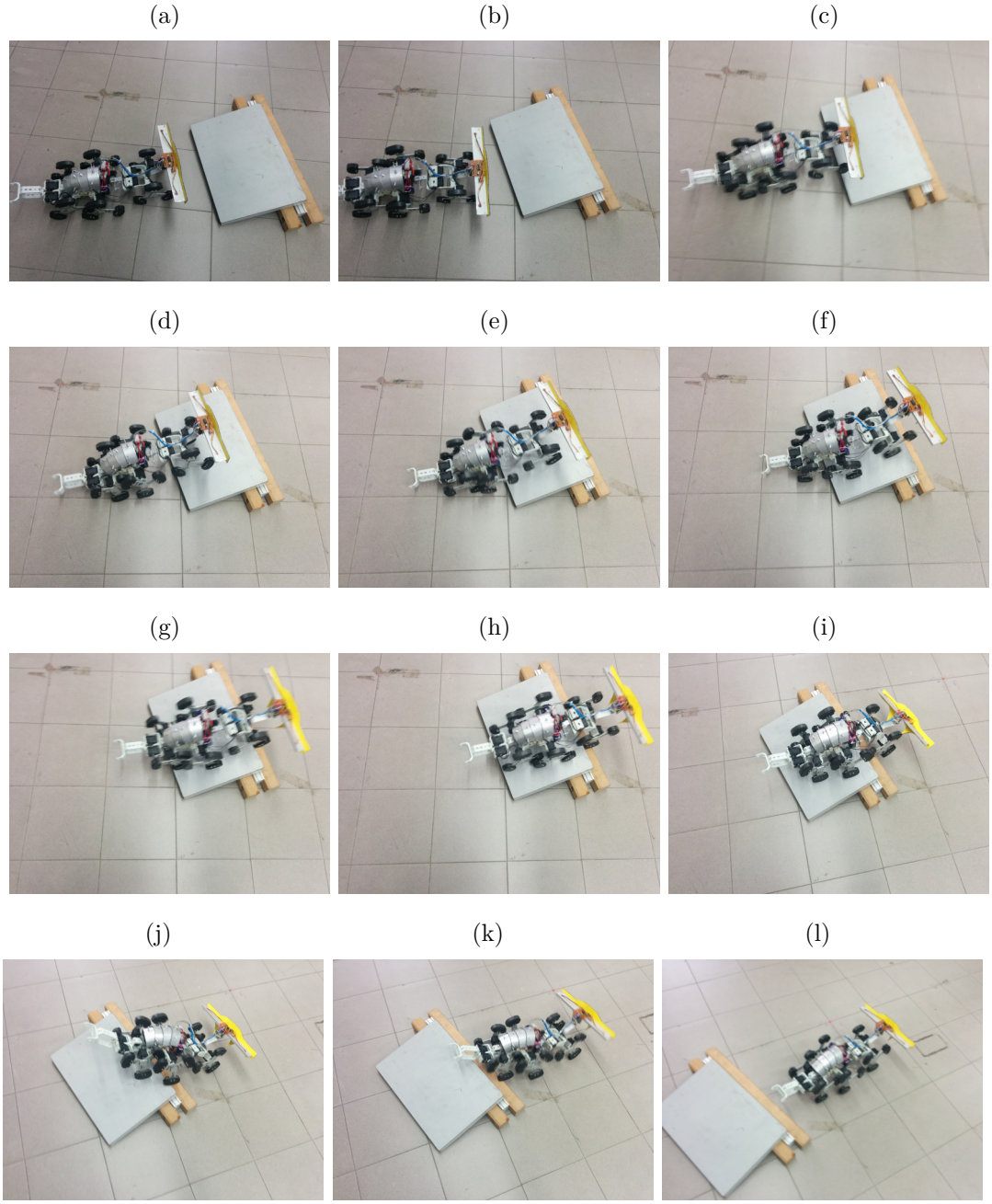


Figura 7.8: LionHell II: rampa, inverso

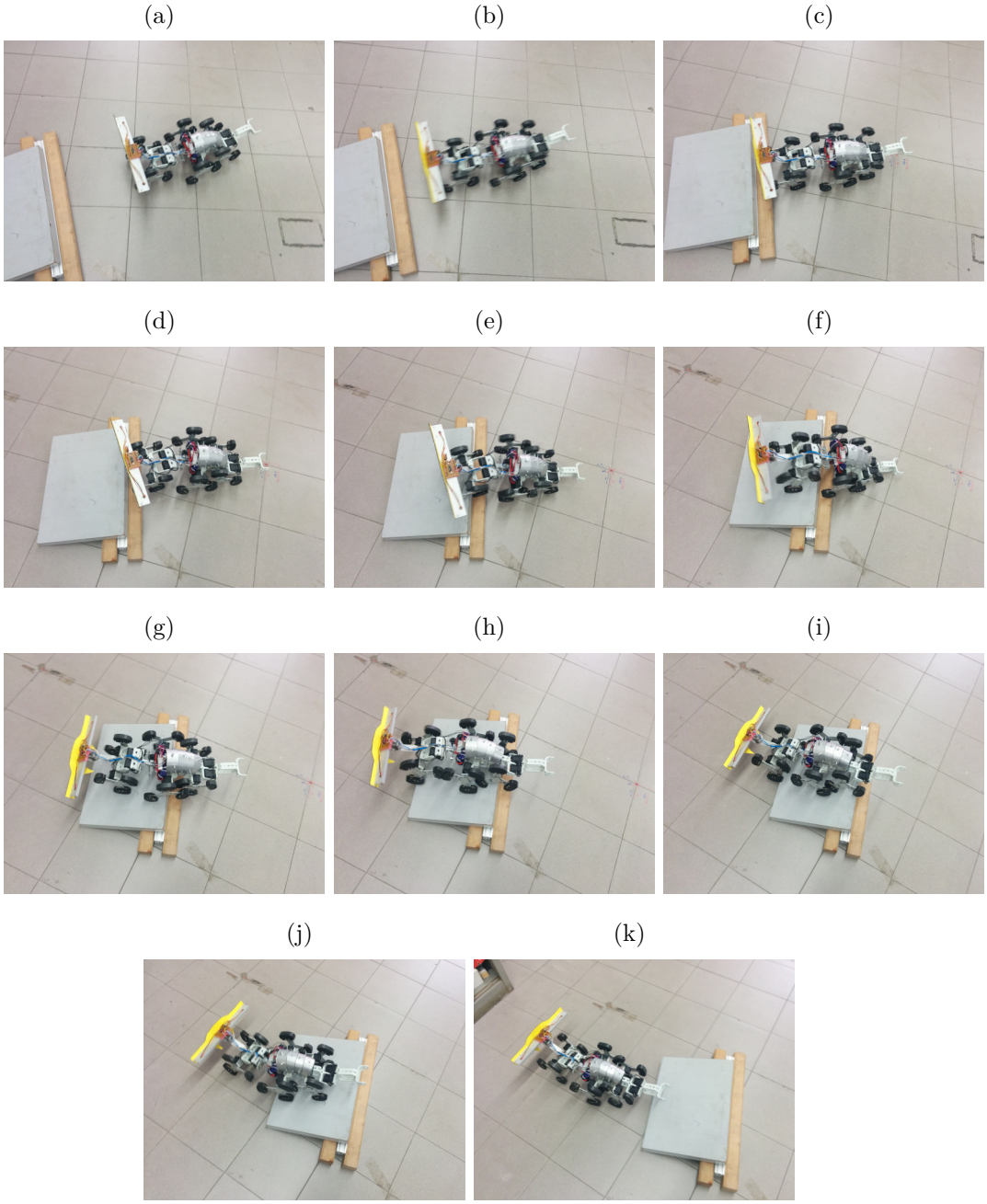


Figura 7.9: LionHell II: tubo

(a) (b) (c)

(d) (e) (f)

(g)

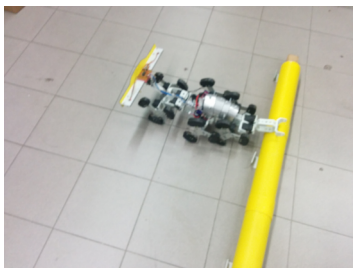
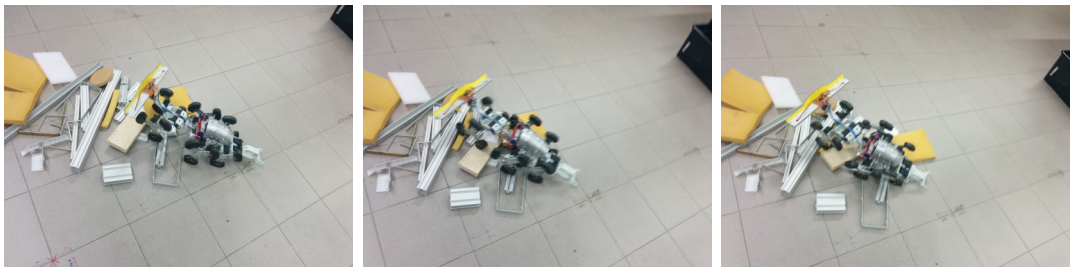


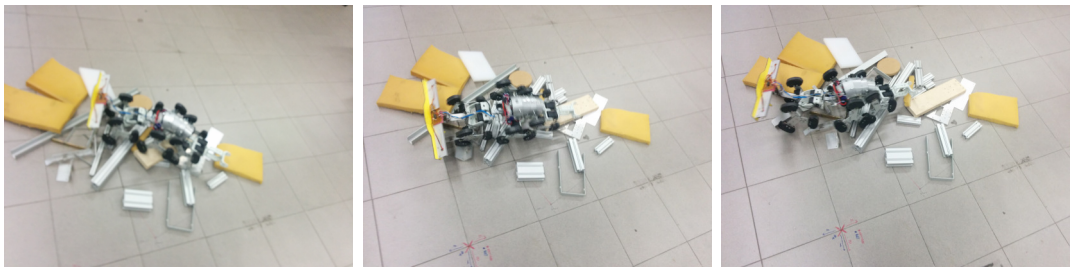
Figura 7.10: LionHell II: terreno cosparso di oggetti

(a) (b) (c)

(d) (e) (f)



(g) (h) (i)



(j) (k) (l)

Bibliografia

[1] Armadillum vulgare (latreille, 1804). http://www.itis.gov/servlet/SingleRpt/SingleRpt?search_topic=TSN&search_value=93250. Accessed: 2015-03-23.

[2] Climbing mini whegs. http://biorobots.case.edu/projects/whegs/cmwhegs/. Accessed: 2015-03-23.

[3] irobot: Our history. http://www.irobot.com/About-iRobot/Company-Information/History.aspx. Accessed: 2015-03-23.

[4] Lorica segmentata o laminata (i sec. a.c. - iii sec. d.c.). http://www.roma-victrix.com/armamentarium/loricae_segmentata.htm. Accessed: 2015-03-23.

[5] Lunar whegs. http://biorobots.case.edu/projects/whegs/lunar-whegs/. Accessed: 2015-03-23.

[6] Panthera tigris ssp. tigris. http://www.iucnredlist.org/details/136899/0. Accessed: 2015-03-23.

[7] Ratasjalg. http://www.roboticsportal.it/it/ratasjalg_robot. Accessed: 2015-03-23.

[8] Stato dei rover marziani alla nasa. http://mars.jpl.nasa.gov/mer/mission/status_opportunityAll.html. Accessed: 2015-03-23.

[9] Usar whegs. http://biorobots.case.edu/projects/whegs/usar-whegs/. Accessed: 2015-03-23.

[10] Vaucanson e i suoi automi. http://www.automates-anciens.com/version_italienne/pagine_principali/automi_vaucanson.htm. Accessed: 2015-03-23.

[11] Whegs, biologically inspired robotics. http://biorobots.case.edu/projects/whegs/. Accessed: 2015-03-23.

[12] Asuilaak living dictionary. http://www.livingdictionary.com/search/viewResults.jsp?language=en&searchString=tupilak&languageSet=all, 2007. Accessed: 2015-03-23.

[13] Robotis e-manual. http://support.robotis.com/en/, 2010. Accessed: 2015-03-23.

[14] Lingua ebraica, in tesauro del nuovo soggettario. http://thes.bncf.firenze.sbn.it/termine.php?id=8620, marzo 2013. Accessed: 2015-03-23.

[15] irobot create open interface (oi) specification. http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf, May 2014. Accessed: 2015-03-23.

[16] R. Altendorfer, N. Moore, H. Komsuoglu, M. Buehler, H.B. Brown, Jr., D. McMordie, U. Saranli, R. Full, and D.E. Koditschek. Rhex: A biologically inspired hexapod runner. *Autonomous Robots*, 11(3):207–213, November 2001.

[17] K. Autumn, S.T. Hsieh, D.M. Dudek, J. Chen, C. Chitaphan, R.J. Full, and author for correspondence rjfull@berkeley.edu. Dynamics of geckos running vertically. *J Exp Biol 209*, pages 260–272, January 2006.

[18] J. Babb, R. Tessier, M. Dahl, S. Hanono, D. Hoki, and A. Agarwal. Logic emulation with virtual wires. http://www.ecs.umass.edu/ece/tessier/courses/697ff/tcad-vw.pdf, 1997. Accessed: 2015-03-23.

[19] J.M. Beer, C. Smarr, T.L. Chen, A. Prakash, T.L. Mitzner, C.C. Kemp, and W.A. Rogers. The domesticated robot: design guidelines for as-

sisting older adults to age in place. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction - HRI '12*, pages 335–342, 2012.

[20] G. Bernardo. Easy bee: Guida alla scelta e alla comprensione dei moduli xbee. http://matteo.luccalug.it/wp-content/uploads/2011/11/easy_bee.pdf, 2011. Accessed: 2015-03-23.

[21] B. Bird. Euphractus sexcinctus six-banded armadillo. http://animaldiversity.org/site/accounts/information/Euphractus_sexcinctus.html, September 2007. Accessed: 2015-03-23.

[22] M. Borgi, I. Cogliati-Dezza, V. Brelsford, K. Meints, and F. Cirulli. Baby schema in human and animal faces induces cuteness perception and gaze allocation in children. *Front Psychol.*, May 2014.

[23] R. Brooks. A robust layered control-system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[24] R. Brooks. I, rodney brooks, am a robot. *IEEE Spectrum*, 48(6):68–75, 2008.

[25] W. Burgard and F. Schneider. Collaborative exploration of unknown environments with teams of mobile robots (2002). In *In dagstuhl*, pages 52–70. Springer Verlag, 2002.

[26] S. Cotton, C. Black, N. Payton, K. Ford, W. Howell, and J. Conrad. Outrunner: The world's first rc running robot. https://www.kickstarter.com/projects/138364285/outrunner-the-worlds-most-advanced-running-robot, July 2014. Accessed: 2015-03-23.

[27] H. Cruse. The control of body position in the stick insect (carausius morosus), when walking over uneven surfaces. *Biological Cybernetics*, 24(1):25–33, March 1976.

[28] A. Cully and J.B. Mouret. Behavioral repertoire learning in robotics. In *GECCO '13 Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 175–182, New York, USA, 2013.

[29] K.A. Daltorio, T.E. Wei, S.N. Gorb, R.E. Ritzmann, and R.D. Quinn. Passive foot design and contact area analysis for climbing mini-whegs. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1274–1279. IEEE, 2007.

[30] K.A. Daltorio, T.C. Witushynsky, G.D. Wile, L.R. Palmer, A.A. Malek, M.R. Ahmad, L. Southard, S.N. Gorb, R.E. Ritzmann, and R.D. Quinn. A body joint improves vertical to horizontal transitions of a wall-climbing robot. In *Robot.,Proceeding of the IEEE International Conference on Intelligent Robots and Systems*, pages 3046–3051, Pasadena, CA, May 2008.

[31] S. Doncieux and J.B. Mouret. Behavioral diversity measures for evolutionary robotics. In *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*, pages 1–8, Barcelona, ES, July 2010. IEEE.

[32] R. Dryden. Study shows humans prefer robots with human faces. http://machinedesign.com/news/study-shows-humans-prefer-robots-human-faces, October 2013. Accessed: 2015-03-23.

[33] P.A. Dunker. A biologically inspired robot for lunar exploration and regolith excavation. Master's thesis, Case Western Reserve University, 2009.

[34] P.A. Dunker, W.A. Lewinger, A.J. Hunt, and R.D. Quinn. A biologically inspired robot for lunar in-situ resource utilization. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 5039–5044, St. Louis, MO, 2009. IEEE.

[35] G. Ehrlich. *This Cold Heaven: Seven Seasons in Greenland*, pages 33–34. Random House, 2001.

[36] C. Filagrossi. Tupilaq. http://ilcrepuscolo.altervista.org/php5/index.php/Tupilaq. Accessed: 2015-03-23.

[37] J. Foster. Xbee cookbook issue 1.4 for series 1 (freescale) with 802.15.4 firmware. http://www.jsjf.demon.co.uk/xbee/xbee.pdf, April 2011. Accessed: 2015-03-23.

[38] A. Gaibotti and F. Mariggiò. Embot : Riprogettazione e realizzazione di un robot esploratore: Meccanica, controllo motori e programmazione. Master's thesis, Politecnico di Milano, 2011.

[39] K.C. Galloway, G.C. Haynes, B.D. Ilhan, A.M. Johnson, R. Knopf, G.A. Lynch, B.N. Plotnick, M. White, and D.E. Koditschek. X-rhex: A highly mobile hexapedal robot for sensorimotor tasks. *Technical Reports (ESE)*, 2010.

[40] D.I. Goldman, T.S. Chen, D.M. Dudek, and R.J. Full. Dynamics of rapid vertical climbing in cockroaches reveals a template. *The Journal of Experimental Biology*, 209:2990–3000, 2006.

[41] L. Greenemeier. What should a robot look like? http://www.scientificamerican.com/article/what-should-a-robot-look-like/, October 2013. Accessed: 2015-03-23.

[42] A.L. Greenfield, U. Saranli, and A. Rizzi. Solving models of controlled dynamic planar rigid-body systems with frictional contact. *The International Journal of Robotics Research*, 24(11):911–931, November 2005.

[43] A.J. Hunt. A biologically inspired robot for assistance in urban search and rescue. Master's thesis, Case Western Reserve University, 2010.

[44] A.J. Hunt, R.J. Bachmann, R.R. Murphy, and R.D. Quinn. A rapidly reconfigurable robot for assistance in urban search and rescue. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 209–214. IEEE, 2011.

[45] M. Idel. *Golem: Jewish Magical and Mystical Traditions on the Artificial Anthropoid*. 1990.

[46] S.R Kellert and E.O. Wilson. The biophilia hypothesis. *Frontiers in Ecology and the Environment*, 5:496, 1993.

[47] H. Kimura and S. Hirose. Development of genbu : Active wheel passive joint articulated mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 823–828, Tokyo, JP, 2002.

[48] W.A. Lewinger. Insect-like antennal sensing for climbing and tunneling behavior in a biologically-inspired mobile robot. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4176–4181. IEEE, April 2005.

[49] V. Lumare. Lionhell mcmillan. http://www.robotgarage.org/wiki/index.php?title=LionHell_McMillan, 1012. Accessed: 2015-03-23.

[50] V. Lumare. Development of a rough terrain wheg robot with morphological computation. Master's thesis, Politecnico di Milano, 2011-2012.

[51] A. Manuel and G. De S. Pablo. Climbing and walking robots. In *Proceedings of the 7th International Conference CLAWAR*, 2004.

[52] A. Martin-Alvarez, P. Peuter, de, J. Hillebrand, P. Putz, A. Matthyssen, and J.F. Weerd, de. Walking robots for planetary exploration missions. In *WAC'96, Second World Automation Congress*, pages 27–30, Montpellier, France, May 1996.

[53] P. Miller. irobot create: Roomba hacking for the everyman. http://www.engadget.com/2006/11/29/irobot-create-roomba-hacking-for-the-everyman/, January 2013. Accessed: 2015-03-23.

[54] M. Mori. The uncanny valley. *Energy*, 7(4):33–35, 1970.

[55] J.M. Morrey, B. Lambrecht, A.D. Horchler, R.E. Ritzmann, and R.D. Quinn. Highly mobile and robust small quadruped robots. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, 2003.

[56] R.P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*, chapter 9, pages 234–244. MIT Press, 1981.

[57] K. Petersen and R. and Nagpal. Termes: An autonomous robotic system for three-dimensional collective construction. In *Robotics: Science and Systems Conference VII*, pages Paper ID 35, Proceedings on CD–ROM, Cambridge, MA, 2011. MIT Press.

[58] R.D. Quinn, G.M. Nelson, R.J. Bachmann, D.A. Kingsley, J. Offi, and R.E. Ritzmann. Insect designs for improved robot mobility. In *Proc. 4th Int. Conf. On Climbing and Walking Robots*, 2001.

[59] U. Saranli, M. Buehler, and D. E. Koditschek. Rhex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7):616–631, 2001.

[60] R. Sell. Ratasjalg. https://www.etis.ee/portaal/isikuCV.aspx?PersonVID=37610&lang=en, June 2007. Accessed: 2015-03-23.

[61] K. Singh. Map making by cooperating mobile robots. In IEEE, editor, *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, volume 2, pages 254–259, Atlanta, May 1993. IEEE.

[62] C.A. Smarr, T.L. Mitzner, J.M. Beer, A. Prakash, T.L. Chen, C.C. Kemp, and W.A. Rogers. Domestic robots for older adults: Attitudes, preferences, and potential. *International Journal of Social Robotics*, 6(2):229–247, 2014.

[63] M. Taddei. *I Robot di Leonardo da Vinci*. Leonardo3, 2007.

[64] M.L. Walters, D.S. Syrdal, K. Dautenhahn, and R.and Koay K.L. Boekhorst, te. Avoiding the uncanny valley - robot appearance, personality and consistency of behavior in an attention-seeking home scenario for

a robot companion. *Autonomous Robots*, 24(Issue 2):159–178, February 2008.

[65] A. Waves. Leg-wheel hybrid for a rover robot: Whegs. https://arunwaves.wordpress.com/2010/07/23/leg-wheel-hybrid-for-a-rover-robot-whegs/, 2010. Accessed: 2015-03-23.

[66] J. Werfel, K. Petersen, and R. Nagpal. Distributed multi-robot algorithms for the termes 3d collective construction system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, Los Angeles, CA, 2011.

[67] J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. *Science*, 343(6172):754–758, 2014.

[68] G.D. Wile, K.A. Daltorio, L.R. Palmer, T.C. Witushynsky, L. Southard, M.R. Ahmad, A. Malek, Ab, S.N. Gorb, A.S. Boxerbaum, R.E. Ritzmann, and R.D. Quinn. Making orthogonal transitions with climbing mini-whegs. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1775–1776, Pasadena, CA, 2008. IEEE.

Appendice A

Listato

```
1  /* AVRGCC1.c
2  *
3  * -- ENGLISH --
4  *
5  * Note: This software is intended to be:
6  *  - compiled with XCODE using AVRGCC on MAC OSX 10.5,
7  *  - compiled with Atmel Studio on Windows 7
8  *  - it can be easily adapted to other compilers on Linux system.
9  *  - executed on Robotis (r) CM-510 Control Board.
10 *    though it can be easily adapted to other Microcontroller Boards
11 *
12 * This code can be downloaded from : http://www.robotgarage.org/wiki/index.php?title=
       LionHell_McMillan
13 *
14 * ----------------------------------------------
15 * Control Logic for LionHell McMillan Rough Terrain Wheg Robot
16 *
17 *
18 * Created by Vittorio Lumare on 10/07/2012.
19 * Modified by Alessandro Rosina on 10/03/2015.
20 * ----------------------------------------------
21 *
22 *
23 Copyright 2012 Vittorio Lumare
24 Copyright 2015 Alessandro Rosina
25 *
26 *
27 Contact: Vittorio Lumare (venom_at_venom_dot_it)
28 Contact: Alessandro Rosina (rosina_at_alessandro_at_0_dot_gmail_at_com)
29 *
30 *
31 This program is free software: you can redistribute it and/or modify  it under the terms
       of the GNU General Public License as published by  the Free Software Foundation,
       either version 3 of the License, or  (at your option) any later version.
32
33 This program is distributed in the hope that it will be useful,  but WITHOUT ANY
      WARRANTY; without even the implied warranty of  MERCHANTABILITY or FITNESS FOR A
      PARTICULAR PURPOSE.  See the  GNU General Public License for more details.
34
35 You should have received a copy of the GNU General Public License  along with this
      program.  If not, see <http://www.gnu.org/licenses/>.
36 *
37 */
38
```

```
39  #include <avr/io.h>
40  #include <util/delay.h>
41  #include <avr/interrupt.h>
42  #include <stdio.h>
43
44  #include "dynamixel.h"
45  #include "serial.h"
46
47  #include <math.h> //libm.a (doesn't work in winavr, works in avr gcc).
48
49  //
50  #define true 1
51  #define false 0
52
53  //Time Vars
54  #define TIME_SEC                        15625
55  #define TIME_DECSEC             1563
56  #define TIME_CENTSEC            156
57  #define TIME_MILLISEC           16
58
59  // AX-12+ Control table addresses
60  #define P_GOAL_POSITION_L               30
61  #define P_GOAL_POSITION_H               31
62  #define P_PRESENT_POSITION_L            36
63  #define P_PRESENT_POSITION_H            37
64  #define P_PRESENT_SPEED_L               38
65  #define P_PRESENT_SPEED_H               39
66  #define P_PRESENT_LOAD_L                40
67  #define P_PRESENT_LOAD_H                41
68  #define P_MOVING                        46
69  #define P_MOVING_SPEED_L                32
70  #define P_MOVING_SPEED_H                33
71  #define P_TORQUE_ENABLE                 24
72
73  // AX S1 Control table addresses
74  #define LEFT_IR_SENSOR_DATA                     26
75  #define CENT_IR_SENSOR_DATA                     27
76  #define RIGH_IR_SENSOR_DATA                     28
77  #define OBSTACLE_DETECTION_FLAG                 32
78  #define OBSTACLE_DETECTED_COMPARE_VALUE         20 //Obstacle Detection Threshold
79  #define OBSTACLE_DETECTED_COMPARE               52 // 0: Low Dist Sens, 1: High Dist
        Sens
80  #define CENTER_LUMINOSITY                       30
81
82  //Button defines
83  #define BTN_UP 0x10
84  #define BTN_DOWN 0x20
85  #define BTN_LEFT 0x40
86  #define BTN_RIGHT 0x80
87  #define BTN_START 0x01
88
89  // Default setting
90  #define DEFAULT_BAUDNUM         1 // 1Mbps
91  #define DEFAULT_ID              1
92
93  struct ServoData{
94          int id;
95          unsigned short lastP[7];//Master Servo: Present Position data
96          _Bool db;//Dead Band state: 0 = not in db; 1 = in db;
97          unsigned short spd; //Master Servo: Present Speed Data
98  };
99
100 //
101 // The structure of the parameters of the IR distance sensors
102 //
103
104 typedef const struct {
105         const signed short a;
```

```
106         const signed short b;
107         const signed short k;
108 }
109 ir_distance_sensor;
110
111 /* All sensor values */
112
113 typedef struct {
114         float distance_front_center; //= readSharp_cm(4);
115         float distance_floor_center; //= readSharp_cm(6);
116         float distance_floor_left; //= readSharp_cm(5);
117         float distance_floor_right; //= readSharp_cm(7);
118         short body_inclination_xg; //= readAccDeg(1);
119         short body_inclination_yg; //= readAccDeg(2);
120         short body_inclination_zg; //= readAccDeg(0);
121         short abdomen_inclination; //= getAbdomenDeg();
122         short abdomen_inclination_g; //= getAbdomenDegG();
123 }
124 Sensors;
125
126
127 void init_OC1A(void);
128 void initTimer1(void);
129 unsigned int Timer1Read( void );
130 void Timer1Write( unsigned int i );
131 void Timer1Pause(unsigned int flag); /*Pause using timer 1 (warning: it resets timer!)
      */
132 void PrintCommStatus(int CommStatus);
133 void PrintErrorCode(void);
134 unsigned char readAXS1cd(void);
135 void go_fwd(void);
136 void stop(void);
137 void torque(unsigned char val);
138 int checkDxlConn(void);
139 void adc_init(void);
140 uint16_t adc_start(unsigned char channel);
141 float ir_distance_calculate_cm(ir_distance_sensor sensor, unsigned short adc_value);
142 unsigned short mirror(unsigned short speed);/* Mirror Speed: 0 to 1023, 1024 to 2047 */
143 unsigned short mirrorPos(unsigned short pos);/* Mirror Position: 0 to 1023 */
144 unsigned short pos2deg(unsigned short pos); /* deg: 0 to 300 */
145 unsigned short deg2pos(unsigned short deg); /* deg: 0 to 300 */
146 short  getTailDeg(void);
147 void setTailDeg(short deg, unsigned short speed);
148 short  getAbdomenDeg(void);/* deg: -75 to +80 */
149 short  getAbdomenDegG(void); /* Gets inclination wrt g-plane. deg: -150 to +150 */
150 void setAbdomenDeg(short deg, unsigned short speed); /* deg: -75 to +80 */
151 unsigned short dist2cm(unsigned short dist);
152 float cosML(float x); float readHeight(void);
153 void setupAcc(void);/* Configure MCU bits to use Accelerometer */
154 uint16_t readAcc(unsigned char addr);
155 float readAccNorm(unsigned char addr);/* read calibrated and normalized (0-1) value from
       Accel Channel @ addr */
156 float readAccDeg(unsigned char addr);/* read deg value from Accel on channel @ addr */
157 void setupSharp(void);/* Configure MCU bits to use Sharp IR Sensors Board */
158 uint16_t readSharp(unsigned char addr); /* Read value of Sharp sensor mapped on address
      "addr" */
159 float readSharp_cm(unsigned char addr); /* Read value of Sharp sensor mapped on address
      "addr"  and convert to cm distance */
160 void readAllSensors(Sensors *s); /* Read all sensor values and put into struct s */
161 void printAllSensors(Sensors sensors);
162 void USART_Flush(void);
163
164 //// Globals ------------------------
165
166 //Communication Dynamixel
167 _Bool comm_err = 0; //Used to ensure correct communication with dynamixel
168 //SHARP Sensor
169 //
```

```
170 // The object of the parameters of GP2Y0A21YK sensor
171 //
172 const ir_distance_sensor GP2Y0A21YK = { 5461, -17, 2 }; //parameters to convert in cm
173
174 //Communication
175 int CommStatus;
176
177 //Whegs, the numbers indicate the ID of each wheg
178 int whegs_sx[3] = {6,3,14};
179 int whegs_dx[3] = {8,7,12};
180
181 //Desired Nominal Speed
182 unsigned short dnspeed = 0;//280;
183
184 struct ServoData sdata[7];
185 unsigned char dCIR; //Center IR Sensor Distance
186 unsigned char dLIR; //Left IR Sensor Distance
187 unsigned char dRIR; //Right IR Sensor Distance
188 unsigned char lCIR; //Center IR Sensor
189 Luminosity _Bool parallel = 1; //Parallel Mode for Whegs
190 int parallel_time = 0; //Parallel Config Time var
191 int frontLift_time = 0; //FrontLift Config Time var
192 int whegBlocked_time = 0; //Wheg Blocked Time var
193 _Bool wheg_blocked = 0 ; //Wheg Blocked state
194 short dpAbd = 0; //Desired Abdomen Inclination wrt Body in [deg]
195 unsigned short pTail ;//Tail Servo Position
196 int dpTail = -1; //-1 : no desidered position (any is good)
197 _Bool dir15 = 0; //sensor servo (id 15) direction. 0 = CW, 1= CCW.
198 Sensors sensors; /* Holds all sensors values */
199 float x_speed = 0; /* X direction speed , 2 be calc'd from x_acc */
200
201 //Button vars
202 _Bool btn_switch1 = 0;
203 _Bool btn_switch2 = 0;
204
205 //Behaviors Controllers
206 unsigned short turnL = 0;
207 unsigned short turnR = 0;
208 _Bool walking = 1;
209 uint16_t resultX, resultY;
210 _Bool bRemoteButton = false;
211
212 int main(void) {
213         serial_initialize(57600);
214         dxl_initialize( 0, DEFAULT_BAUDNUM ); // Not using device index
215         sei(); // Interrupt Enable
216         printf( "\n\n Venom Whegs Sync Master\n\n" );
217         _delay_ms(20000); // Enable Continuous Turn on whegs servos
218         int i;
219         for (i=0;i<3;i++){
220                 // dxl_write_word(<servo number>, <addresses>, <value>)
221                 dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L, 0 ); // set whegs_sx[i]
                        moving speed to 0 position
222                 dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L, 1024 ); // set whegs_dx[i
                        ] moving speed to 1024 position, inverting the direction of movement
223                 dxl_write_word( whegs_sx[i], 8, 0 ); // set whegs_sx[i] CCW_ANGLE_LIM to
                        0
224                 dxl_write_word( whegs_dx[i], 8, 0 );
225         }
226
227         //Disable Continuous Turn for Body Torsion Servo and Tail Servo
228         dxl_write_word( 10, 8, 1023 ); //Disable Continuous Turn Mode
229         dxl_write_word( 10, P_TORQUE_ENABLE, 0 ); //Disable Torsion Servo
230         dxl_write_word( 1, 8, 1023 ); //Disable Continuous Turn Mode
231         dxl_write_word( 1, P_TORQUE_ENABLE, 0 ); //Disable Tail Servo
232         // Write moving speed
233         //dxl_write_word( 6, P_MOVING_SPEED_L, 150 );
234         //dxl_write_word( 8, P_MOVING_SPEED_L, 1174 );
```

```
235
236         //Prepare servos data
237         sdata[0].id = whegs_sx[0];
238         sdata[1].id = whegs_sx[1];
239         sdata[2].id = whegs_sx[2];
240         sdata[3].id = whegs_dx[0];
241         sdata[4].id = whegs_dx[1];
242         sdata[5].id = whegs_dx[2];
243         sdata[6].id = 15; //Distance Sensor Servo
244
245         dxl_write_word(15, 8, 1023); //Position Mode for Sensor Servo
246
247         //Remote Button Init
248         DDRE  = 0x0C;
249         PORTE = 0xF0;
250         //DDRE |= (0<<BTN_RIGHT) ; //MCU PIN BTN_RIGHT set as Input (button has been
                physically removed)
251
252         //Prepare Sensor
253         dxl_write_byte( 100, OBSTACLE_DETECTED_COMPARE, 1 );
254         //High Dist Sens
255         //dxl_write_byte(100,OBSTACLE_DETECTED_COMPARE_VALUE, 50); //50 is detection
                threshold
256
257         adc_init(); //This function initializes the ADC hardware and asynchronous ADC
                read subsystem
258         setupAcc();
259         setupSharp();
260
261         initTimer1();
262         walking = false;
263         USART_Flush();
264
265         while(1) {
266                 // Read Remote Controller via XBee using Virtual Wires {
267                         resultX = adc_start ( 4 );
268                         resultY = adc_start ( 3 );
269                         bRemoteButton = (PINE & BTN_RIGHT);
270                         //printf("\r\nresultX resultY button : %u %u %d", resultX,
                                resultY, bRemoteButton);
271
272                         // BUTTON
273                         if (bRemoteButton) {
274                                 walking = true;
275                         }
276                         else
277                         {
278                                 walking = false;
279                         }
280
281                         //X
282                         if ( resultX > 340 ) {
283                                 turnL = 0; turnR=0; // Go Fwd
284                         }
285                         else if ( resultX < 300  ) {
286                                 turnL = 1; turnR=1; // Go Bwd
287                         }
288
289                         //Y
290                         if ( resultY > 340 ) {
291                                 turnR = 1; turnL=0; // Turn Right
292                         }
293                         else if ( resultY < 300 ) {
294                                 turnL = 1; turnR=0; // Turn Left
295                         }
296                 }
297
298                 //Read and Communicate Sensors Data
```

```
299                 readAllSensors(&sensors);
300                 printAllSensors(sensors);
301
302                 // Set Speed According to Body Inclination
303                 dnspeed = (600 + 4 *sensors.body_inclination_xg); //set speed according
                        to terrain inclination(more speed = more force)
304
305                 //printf("\r\nturnL turnR walking : %u %u %d %d", turnL, turnR, walking,
                        dnspeed);
306                 //Main Behaviors Section
307                 //Vars for Behaviors
308                 float ld = abs((int)(sensors.distance_floor_left - sensors.
                        distance_floor_center));
309                 float rd = abs((int)(sensors.distance_floor_right - sensors.
                        distance_floor_center));
310
311                 //Jump to Floor Behavior
312                 if (sensors.abdomen_inclination_g < -30 && sensors.distance_front_center
                        < 30){ //..isFalling, TerrainIsNear -> JUMP!
313                     stop();
314                     comm_err = 0; //communication error
315                     while(sensors.abdomen_inclination_g < -30 && sensors.
                            distance_front_center < 30){ //JUMP
316                         //printf("\nJUMP\n");
317                         if (comm_err == 0)
318                                 dpAbd = dpAbd + 1;
319                         ERROR //LIFT BODY
320                         comm_err = 0; //reset communication error
321                         //setTailDeg(90, 1023); //Lift tail at super speed (back
                                body goes down, front body goes up)
322                         setAbdomenDeg(dpAbd, 1023);//Lift
323                         if (checkDxlConn()==-1)
324                                 comm_err = 1;
325                         else{
326                                 int isMoving = 1;
327                                 while(isMoving == 1) {//reach pos
328                                         isMoving = dxl_read_byte(16,P_MOVING);
329                                         if (checkDxlConn()==-1){
330                                                 comm_err = 1;
331                                                 isMoving = 0; //permits exiting
                                                        from while
332                                         }
333                                 }
334                         }
335                         sensors.abdomen_inclination_g = getAbdomenDegG();
336                         sensors.distance_front_center = readSharp_cm(4);
337                     }
338                     //setTailDeg(0, 512);
339                     //put tail in horizontal pos
340                     //while(dxl_read_byte(1,P_MOVING)); //wait pos reached
341                     //dxl_write_word( 1, P_TORQUE_ENABLE, 0 );
342                     //Disable Tail Servo
343                 }else
344                 //Terrain Check Behavior (DISABLE BECAUSE VISION SYSTEM IS INSUFFICIENT)
345                 if( 0 && (ld > 8 || rd > 8)) { //terrain not uniform //last=4
346                     //printf("ld:%d,rd:%d\n",(int)ld,(int)rd);
347                     if (ld >= rd) {
348                         turnR = 1; turnL=0;
349                     }
350                     if (ld < rd) {
351                         turnL = 1; turnR=0;
352                     }
353                 }
354                 else if (sensors.distance_front_center <  15){
355                     //Near Center Obstacle..
356                     stop(); //STOP WALKING
357                     //Approaching Obstacle Behavior
358                     comm_err = 0; //communication error
```

```
359                     while(sensors.distance_front_center < 15 && sensors.
                            abdomen_inclination < 80){//LIFT UP..
360                         //printf("\nLIFT UP TO CLIMB\n");
361                         if (comm_err == 0 )dpAbd = dpAbd + 1;
362                         if (dpAbd > 80)dpAbd = 80;
363                         //LIFT BODY
364                         comm_err = 0; //reset communication error
365                         setAbdomenDeg(dpAbd, 200); //Lift
366                         if (checkDxlConn()==-1)
367                                 comm_err = 1;
368                         else{
369                                 int isMoving = 1;
370                                 while(isMoving == 1) { //reach pos
371                                         isMoving = dxl_read_byte(16,P_MOVING);
372                                                 if (checkDxlConn()==-1){
373                                                         comm_err = 1;
374                                                         isMoving = 0; //permits
                                                                exiting from while
375                                                 }
376                                         }
377                                 }
378                                 sensors.distance_front_center = readSharp_cm(4);
379                         }
380                     }
381                     else //Adapt to Floor Behavior
382                     if(sensors.distance_floor_center > 23 ){ // Floor too distant ..
383                             stop();
384                             comm_err = 0;
385                             //communication error
386                             while(sensors.distance_floor_center > 23 && sensors.
                                    abdomen_inclination > -75){ // LOWER DOWN..
387                                 //printf("\nLOWERING TO THE FLOOR\n");
388                                 if (comm_err == 0 )
389                                         dpAbd = dpAbd - 1;
390                                 if (dpAbd < -75)
391                                         dpAbd = -75;
392                                 comm_err = 0; //reset communication error
393                             setAbdomenDeg(dpAbd, 200); //Lower Abdomen
394                             if (checkDxlConn()==-1)
395                                     comm_err = 1;
396                             else{
397                                     int isMoving = 1;
398                                     while(isMoving == 1) { //reach pos
399                                             isMoving = dxl_read_byte(16,P_MOVING);
400                                             if (checkDxlConn()==-1){
401                                                     comm_err = 1;
402                                                     isMoving = 0; //permits exiting
                                                            from while
403                                             }
404                                     }
405                             }
406                             sensors.distance_floor_center = readSharp_cm(6);
407                     }
408                 }
409                 //End of Behaviors
410                 //Walking Actions
411                 if (walking==1){
412                         if(turnL && turnR){
413                             int i ;
414                             for (i=0;i<3;i++){ //Go Backward
415                                     dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L,
                                            1624 );
416                                     dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L,
                                            600 );
417
418                                     //--START (DISABLE)
419                                     //_delay_ms(40000); //Mantain behavior
420                                     //Disable behavior..
```

```
421                                     //turnL = 0;
422                                     //turnR = 0;
423                                     //--END
424                             }
425                     }else if(turnL){ //Turn Left
426                             int i ;
427                             for (i=0;i<3;i++){
428                                     dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L,
                                            1624 );
429                                     dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L,
                                            1624 );
430                                     //dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L,
                                             0 );
431
432                                     //--START (DISABLE)
433                                     //_delay_ms(20000); //Mantain behavior
434                                     //Disable behavior..
435                                     //turnL = 0;
436                                     //turnR = 0;
437                                     //--END
438                             }
439                     }else
440                     if(turnR){ //Turn Right
441                             int i ;
442                             for (i=0;i<3;i++){
443                                     dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L,
                                            600 );
444                                     //dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L,
                                             0 );
445                                     dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L,
                                            600 );
446
447                                     //--START (DISABLE)
448                                     //_delay_ms(20000); //Mantain behavior
449                                     //Disable behavior..
450                                     //turnL = 0;
451                                     //turnR = 0;
452                                     //--END
453                             }
454                     }else if (!(turnL + turnR)){ //If not turning
455                             go_fwd(); //Restart walking
456                     }
457             }else{ //Stop Walking
458                     stop();
459             }
460
461             //Tail Control Section
462             pTail = (unsigned short)dxl_read_word( 1, P_PRESENT_POSITION_L);
463             //printf("%d ",pTail);
464             //Tail Behaviors Manager
465             //Check if desidered Tail position (if was set) has been reached
466             if(dpTail != -1 && abs((int)dpTail - (int)pTail) < 50){
467                     dxl_write_word( 1, P_TORQUE_ENABLE, 0 ); //Disable Tail Servo
468                     // Needed to shut off tail behaviors
469                     // activated in previous loop cycle
470                     dpTail = -1; //disable desidered Tail position (any will be good
                            )
471             }
472
473             //Tail Behavior 1: Avoiding Falling Backward
474             //If very high tail and abdomen not low and body tilted up
475             if (parallel && pTail < 300 && getAbdomenDeg() >-10 && readAccDeg(1) >
                    45){//avoid falling backward
476                     //printf("\navoid falling backward\n");
477                     dpTail = 500;
478                     dxl_write_word( 1, P_GOAL_POSITION_L, dpTail );
479                     dxl_write_word( 1, P_MOVING_SPEED_L, 210 );
480             }
```

```
481
482                     //Tail Behavior 2: Climbing
483                     //If quite high tail and abdomen quite low and body tilted up
484                     if (parallel && pTail < 450 && getAbdomenDeg() < -20 && readAccDeg(1) >
                            60 ){//climb
485                             //printf("\nclimb\n");
486                             dpTail = 712;
487                             dxl_write_word( 1, P_GOAL_POSITION_L, dpTail );
488                             dxl_write_word( 1, P_MOVING_SPEED_L, 210 );
489                     }else
490
491                     //Working on timer.. //
492                     time = Timer1Read(); //
493                     //printf(" %u ", time/(unsigned int)TIME_MILLISEC);
494                     //printf(" %u ", time);
495                     //printf(" main ");
496                     //printf("\n");
497
498                     if(0){//STOP MOTION
499                             //printf("PRESS KEY\n");
500                             torque(0);
501                             unsigned char ReceivedData = getchar();
502                             torque(1);
503                     }
504             }       //main loop
505             return 0;
506     }
507
508     void printAllSensors(Sensors sensors){
509             //printf("dnspeed:\t%d\t",dnspeed);
510             printf("FC %d\t",(int)(sensors.distance_front_center));
511             printf("GC %d\t",(int)(sensors.distance_floor_center));
512             printf("GL %d\t",(int)(sensors.distance_floor_left));
513             printf("GR %d\t",(int)(sensors.distance_floor_right));
514
515             printf("Z %d\t",(short)(1000*readAccNorm(0)));
516             printf("X %d\t",(short)(1000*readAccNorm(1)));
517             printf("Y %d\t",(short)(1000*readAccNorm(2)));
518
519             printf("AD %d\t",getAbdomenDeg());
520             printf("TD %d\n",getTailDeg());
521     }
522
523     // Print communication result
524     void PrintCommStatus(int CommStatus) {
525             switch(CommStatus)      {
526                     case COMM_TXFAIL:
527                             printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
528                             break;
529                     case COMM_TXERROR:
530                             printf("COMM_TXERROR: Incorrect instruction packet!\n");
531                             break;
532                     case COMM_RXFAIL:
533                             printf("COMM_RXFAIL: Failed get status packet from device!\n");
534                             break;
535                     case COMM_RXWAITING:
536                             printf("COMM_RXWAITING: Now recieving status packet!\n");
537                             break;
538                     case COMM_RXTIMEOUT:
539                             printf("COMM_RXTIMEOUT: There is no status packet!\n");
540                             break;
541                     case COMM_RXCORRUPT:
542                             printf("COMM_RXCORRUPT: Incorrect status packet!\n");
543                             break;
544                     default:
545                             printf("This is unknown error code!\n");
546                             break;
547             }
```

```
548 }
549
550 // Print error bit of status packet void PrintErrorCode() {
551         if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
552                 printf("Input voltage error!\n");
553         if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
554                 printf("Angle limit error!\n");
555         if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
556                 printf("Overheat error!\n");
557         if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
558                 printf("Out of range error!\n");
559         if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
560                 printf("Checksum error!\n");
561         if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
562                 printf("Overload error!\n");
563         if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
564                 printf("Instruction code error!\n");
565 }
566
567 unsigned int Timer1Read( void ) {
568         unsigned char sreg;
569         unsigned int i;
570         /* Save global interrupt flag */
571         sreg = SREG;
572         /* Disable interrupts */
573         cli() ; //__disable_interrupt();
574         /* Read TCNTn into i */
575         i = TCNT1;
576         /* Restore global interrupt flag */
577         SREG = sreg;
578         return i;
579 }
580
581 void Timer1Write( unsigned int i ) {
582         unsigned char sreg;
583         /* Save global interrupt flag */
584         sreg = SREG;
585         /* Disable interrupts */
586         cli(); //__disable_interrupt();
587         /* Set TCNTn to i */
588         TCNT1 = i;
589         /* Restore global interrupt flag */
590         SREG = sreg;
591 }
592
593 void Timer1Pause(unsigned int flag){
594         /*Pause using timer 1 */
595         Timer1Write(0);
596         unsigned int time = Timer1Read();
597         while(time < flag ){
598                 time = Timer1Read();
599         }
600 }
601
602 void initTimer1(void){
603         TCCR1B = (1<<CS12)|(1<<CS10); //Prescaler to Sys Clock / 1024 = (16MHz / 1024)
604 }
605
606 void init_OC1A(void) {
607         //ASSR= 1<<AS2; // Enable asynchronous mode
608         TCCR1B = (1<<CS12)|(1<<CS10);//Prescaler to Sys Clock / 1024 = (16MHz / 1024)
609         TIFR1 = 1<<OCF1A; // Clear OCF1/ Clear pending interrupts
610         TIMSK1= 1<<OCIE1A; // 1:Enable (0:Disable) Timer1 Output Compare Mode Compare
                Match Interrupt
611         OCR1A= 48; // Set Output Compare Value to 32
612         //Timer1Write((unsigned int) 0);//Resets Timer
613         //DDRB= 0xFF; // Set Port B as output
614         //while (ASSR&(1<<OCR2UB)); // Wait for registers to update
```

```
615 }
616
617 //Interrupt Function
618
619 ISR(TIMER1_COMPA_vect){
620         // Clear timer on compare match (now manually)
621         Timer1Write((unsigned int) 0);//Resets Timer
622         float deltaT = 1000/dnspeed; //msecs per count (AX12 has 1023 counts for 300 deg
                )
623         OCR1A = TIME_MILLISEC * deltaT;// set compare interval
624 }
625
626 unsigned short mirror(unsigned short speed){//calc specular speed
627         if (speed > 1023)
628                 return speed - 1024;
629         else
630                 return speed + 1024;
631 }
632
633 void adc_init(void){ // Function used to initialise the ADC feature
634         ADMUX=0X40; // 0x40 for 10bit //0x60 for 8bit
635         ADCSRA=0X87; // We have set the ADSC bit to start a conversion, and the
636         // ADPS bits are set so that the prescaler is 128
637         //ADCSRA=0X80;
638         // ADEN is set, to enable the ADC
639 }
640
641 uint16_t adc_start(unsigned char channel){ // Function to perform an ADC conversion,
      Takes 0-8 as input
642         // to select which input to convert
643         unsigned char i;
644         i=channel&0x07; // Decide which line to perform ADC conversion on
645         //ADMUX=i|0x60; // Enter which line to perform in the ADC control register (for
                8 bit)
646         ADMUX=i|0x40; // Enter which line to perform in the ADC control register (for 10
                bit)
647         ADCSRA|=1<<ADSC;
648
649         while(ADCSRA & (1<<ADSC));  // wait for conv. to complete
650         //unsigned char temp=ADCH; //for 8-bit
651         unsigned short temp;
652         temp=ADC;//for 10-bit
653         return temp;
654 }
655
656 int checkDxlConn(void){ //checkDxlConn get the result and if it has success it prints
      the status, if not it returns -1
657         CommStatus = dxl_get_result();
658         if( CommStatus != COMM_RXSUCCESS ){
659                 PrintCommStatus (CommStatus);
660                 PrintErrorCode();
661                 return -1;
662         }
663         return 1;
664 }
665
666 unsigned short mirrorPos(unsigned short pos){
667         /* Calc pos for upside-down servo */
668         pos = 1023 - pos;
669         return pos;
670 }
671
672 unsigned short pos2deg(unsigned short pos){
673         /* deg : 0 to 300 . 150 is center */
674         if(pos > 1023)return -1;
675         /* Error: out of range */
676         //printf(" p%u ",pos);
677         uint32_t deg;
```

```
678         //unsigned int deg;
679         deg = (uint32_t)pos;
680         //printf(" d%u ",deg);
681         deg = deg * 293; //it was 29.3
682         //printf(" d%u ",deg);
683         deg = deg / 1000;
684         //printf(" d%u ",deg);
685         return (unsigned short)deg;
686 }
687
688 unsigned short deg2pos(unsigned short deg){
689         /* deg : 0 to 300 . 150 is center */
690         if (deg > 300) return -1 ;
691         /* Error: out of range */
692         //printf(" p%u ",deg);
693         uint32_t pos;
694         //unsigned int deg;
695         pos = (uint32_t)deg;
696         //printf(" d%u ",pos);
697         pos = pos * 1000;
698         //printf(" d%u ",pos);
699         pos = pos / 293; //it was 29.3
700         //printf(" d%u ",pos);
701         return (unsigned short)pos;
702 }
703
704 short  getTailDeg(void){
705         short deg = (short)pos2deg(dxl_read_word(1, P_PRESENT_POSITION_L)) - 150;
706         return  -deg; //servo is mounted upside-down
707 }
708
709 void setTailDeg(short deg, unsigned short speed){
710         deg = - deg; //servo is mounted upside-down
711         if (deg < -90 || deg > 90 ){
712                 /* Error: out of range */
713                 printf("\nError: deg must be in the -90 to 90 deg range.\n");
714         }
715         else{
716                 unsigned short posTail = -1; //holds abdomen joint position
717                 posTail = deg2pos((unsigned short)(deg + 150));
718                 //printf ("1pos:\t%d\t",posTail);
719                 dxl_write_word(1, P_MOVING_SPEED_L, speed);
720                 dxl_write_word(1, P_GOAL_POSITION_L, posTail);
721                 posTail = mirrorPos(posTail ) ;
722                 //printf ("5pos:\t%d\n",posTail);
723                 dxl_write_word(5, P_MOVING_SPEED_L, speed);
724                 dxl_write_word(5, P_GOAL_POSITION_L, posTail);
725         }
726 }
727
728 short  getAbdomenDeg(void){
729         return (short)pos2deg(dxl_read_word(16, P_PRESENT_POSITION_L)) - 150;
730 }
731
732 short  getAbdomenDegG(void){ /* Gets inclination wrt g-plane */
733         return getAbdomenDeg() + (short)readAccDeg(1);
734 }
735
736 void setAbdomenDeg(short deg, unsigned short speed){
737         if (deg < -75 || deg > 80 ){
738                 /* Error: out of range */
739                 printf("\nError: deg must be in the -75 to + 80 deg range.\n");
740         }
741         else{
742                 unsigned short posAbdomen = -1; //holds abdomen joint position
743                 posAbdomen = deg2pos((unsigned short)(deg + 150));
744                 //printf ("16pos:\t%d\t",posAbdomen);
745                 dxl_write_word(16, P_MOVING_SPEED_L, speed);
```

```
746                     dxl_write_word(16, P_GOAL_POSITION_L, posAbdomen);
747                     posAbdomen = mirrorPos(posAbdomen ) ;
748                     //printf ("18pos:\t%d\n",posAbdomen);
749                     dxl_write_word(18, P_MOVING_SPEED_L, speed);
750                     dxl_write_word(18, P_GOAL_POSITION_L, posAbdomen);
751             }
752             unsigned short dist2cm(unsigned short dist){//Convert distance from SHARP
                    GP2Y0A21YK sensor to cm
753                     uint32_t cm = (uint32_t)dist;
754                     cm = 12344 * cm ;
755                     return cm;
756             }
757     }
758
759     //
760     // Converting the values of the IR distance sensor to centimeters
761     // Returns -1, if the conversion did not succeed
762     //
763
764     float ir_distance_calculate_cm(ir_distance_sensor sensor, unsigned short adc_value){
765             if (adc_value + sensor.b <= 0) {
766                     return -1;
767             }
768             float dist_cm;
769             dist_cm = (float)sensor.a / (float)(adc_value + sensor.b) - (float)sensor.k;
770             //dist_cm =  5461 / (adc_value + -17) - 2;
771             //printf("cm%d ",dist_cm);
772             return dist_cm;
773     }
774
775     float cosML(float x){ //cosine MacLaurin 4 term  (GOOD from 0 to 45 ,need to improve to
            50)
776             //deg 2 rad
777             x = x * 0.017453;
778             //printf("r%d ",(int)(x*10));
779             //cos(x):  1-0.5x^2 +  0.04167 * x^4
780             float cosine;
781             cosine = 1- 0.5 * x*x + 0.04167 * x *x *x *x ;
782             //printf("cos%d ",(int)(cosine*10));
783             return cosine;
784     }
785
786     float readHeight(void){
787             //Read Sensor
788             uint16_t result=adc_start(6);
789             // cHANNEL 6
790             //printf("r%u ", result);
791             float dist_cm;
792             dist_cm = ir_distance_calculate_cm(GP2Y0A21YK, result);
793             //Convert
794             //printf("cm%d ", (int)(dist_cm*10));
795             float height ;
796             height = 4.5 + (float)dist_cm;
797             //Add sensor stand length to Height
798             if(height<10)
799                     height = 10;
800             else if(height>80)
801                     height = 80;
802             return height ;
803     }
804
805     void go_fwd(void){
806             printf ( "\r\n GO FWD dnspeed=%d", dnspeed);
807             int i;
808             for (i=0;i<3;i++)
809                     dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, dnspeed );
810             for (i=3;i<6;i++)
811                     dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, mirror(dnspeed ));
```

```
812 }
813
814 void stop(void){
815         int i ;
816         for (i=0;i<3;i++)
817                 dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, 0 );
818         for (i=3;i<6;i++)
819                 dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, mirror(0));
820 }
821
822 void torque(unsigned char val){
823         int i ;
824         for (i=0;i<3;i++)
825                 dxl_write_word( sdata[i].id, P_TORQUE_ENABLE, val );
826         for (i=3;i<6;i++)
827                 dxl_write_word( sdata[i].id, P_TORQUE_ENABLE, val);
828 }
829
830 unsigned char readAXS1cd(void){ //Read AX-S1 Sensor Center Distance Filtering Errors and
        Compensating Ambient Light
831         int i,dCIR,lCIR,cdCIR,dCIRmin; //corrected distance
832         dCIRmin = 255;
833         for(i=0;i<10;i++){ //Collect min val (avoid ambient ir-modulated interferences )
834                 do{dCIR = (unsigned char) dxl_read_byte( 100, CENT_IR_SENSOR_DATA);}
835                         while (dxl_get_result()!=COMM_RXSUCCESS);
836                 if (dCIR < dCIRmin)
837                         dCIRmin = dCIR;
838         } //using dCIRmin as distance value
839         do{lCIR = (unsigned char) dxl_read_byte(100,CENTER_LUMINOSITY);}
840                 while (dxl_get_result()!=COMM_RXSUCCESS);
841         cdCIR = dCIRmin - 2*lCIR ; //Compensate Ambient Luminosity
842         if(cdCIR <0)cdCIR = 0; //Lower bound
843         //printf("dCIRmin %d \t",dCIRmin);
844         //printf("lCIR %d \t",lCIR);
845         //printf("cdCIR %d \n",cdCIR);
846         //if(dCIRmin > 100)printf(" >100 \n");
847                 //return cdCIR;
848 }
849
850 void setupAcc(void){
851         /* Configure MCU bits to use Accelerometer */
852         //Configure MCU outputs to interface Acc inputs
853         DDRA |= (1<<PA7)|(1<<PA6) ; //MCU PIN A6 and A7 set as Output
854 }
855
856 uint16_t readAcc(unsigned char addr){
857         /* Note: All PORTA pin values are inverted (1 is LOW, 0 is HIGH)
858         * This is due to cm-510 output transistor, I guess
859         */
860         switch (addr){
861                 case 0 :
862                         // Choosing Z Accel.
863                         PORTA = (1<<PA7)|(1<<PA6); //MPLEX: ADDR 00 (0)
864                         //printf("\n00:Z \t");
865                         break;
866                 case 1 :
867                         // Choosing X Accel.
868                         PORTA = (1<<PA7)|(0<<PA6); //MPLEX: ADDR 01 (1)
869                         //printf("\n01:X \t");
870                         break;
871                 case 2 :
872                         // Choosing Y Accel.
873                         PORTA = (0<<PA7)|(1<<PA6); //MPLEX: ADDR 10 (2)
874                         //printf("\n10:Y\t");
875                         break;
876                 case 3 :
877                         // Chosing Free Address
878                         PORTA = (0<<PA7)|(0<<PA6); //MPLEX: ADDR 11 (3)
```

```
879                             //printf("\n11:F\t");
880                             break;
881                     default:
882                             printf("\nWrong Address (%d) Selected on channel 2\n", addr);
883                             return -1;
884             }
885             //Pause before acquiring..
886             Timer1Write(0);
887             int time = Timer1Read();
888             while(time < TIME_MILLISEC){
889                     time = Timer1Read();
890                     //printf("time=%d\n",time);
891             }
892             int16_t result=adc_start(2);
893             //Channel 2
894             //printf("result : \t%d\n",result);
895             return result;
896     }
897
898     void setupSharp(void){
899             /* Configure MCU bits to use Sharp IR Sensors Board */
900             //Configure MCU outputs to interface Board inputs
901             DDRA |= (1<<PA2)|(1<<PA3);
902             //MCU PIN PA2 and PA3 set as Output
903             DDRF |= (1<<PF5) ; //MCU PIN PF4 set as Output
904     }
905
906     uint16_t readSharp(unsigned char addr){
907             /* Read value of Sharp sensor mapped on address "addr" */
908             /* Note: All PORTA pin values are inverted (1 is LOW, 0 is HIGH)
909             * This is due to cm-510 output transistor, I guess
910             */
911             switch(addr){
912                     case 0:
913                             //MPLEX: ADDR 000 (0)
914                             PORTA = (1<<PA3)|(1<<PA2); //BIT1:0 = 00
915                             PORTF = (0<<PF5);//BIT2 = 0
916                             break;
917                     case 1:
918                             //MPLEX: ADDR 001 (1)
919                             PORTA = (1<<PA3)|(0<<PA2); //BIT1:0 = 01
920                             PORTF = (0<<PF5);//BIT2 = 0
921                             break;
922                     case 2:
923                             //MPLEX: ADDR 010 (2)
924                             PORTA = (0<<PA3)|(1<<PA2); //BIT1:0 = 10
925                             PORTF = (0<<PF5);//BIT2 = 0
926                             break;
927                     case 3:
928                             //MPLEX: ADDR 011 (3)
929                             PORTA = (0<<PA3)|(0<<PA2); //BIT1:0 = 11
930                             PORTF = (0<<PF5);//BIT2 = 0
931                             break;
932                     case 4:
933                             //MPLEX: ADDR 100 (4)
934                             PORTA = (1<<PA3)|(1<<PA2); //BIT1:0 = 00
935                             PORTF = (1<<PF5);//BIT2 = 1
936                             break;
937                     case 5:
938                             //MPLEX: ADDR 101 (5)
939                             PORTA = (1<<PA3)|(0<<PA2); //BIT1:0 = 01
940                             PORTF = (1<<PF5);//BIT2 = 1
941                             break;
942                     case 6:
943                             //MPLEX: ADDR 110 (6)
944                             PORTA = (0<<PA3)|(1<<PA2); //BIT1:0 = 10
945                             PORTF = (1<<PF5);//BIT2 = 1
946                             break;
```

```
947                     case 7:
948                             //MPLEX: ADDR 111 (7)
949                             PORTA = (0<<PA3)|(0<<PA2); //BIT1:0 = 11
950                             PORTF = (1<<PF5);//BIT2 = 1
951                             break;
952                     default :
953                             printf("\nWrong Address (%d) Selected on channel 2\n", addr);
954                             return -1;
955             }
956             //Pause before acquiring..
957             Timer1Write(0);
958             int time = Timer1Read();
959             while(time < TIME_MILLISEC ){
960                     time = Timer1Read();
961                     //printf("time=%d\n",time);
962             }
963             int16_t result = adc_start(6);
964             //Channel 6
965             //filtering : pass only 0.3V (64) - 3V (613)
966             if (result < 64)
967                     result = 64;
968             if (result > 613)
969                     result = 613;
970             //printf("result : \t%d\n",result);
971                     return result;
972     }
973
974     unsigned short sharp_calib[17][2]={
975             {63,40},
976             {71,35},
977             {85,30},
978             {106,25},
979             {132,20},
980             {153,18},
981             {165,16},
982             {192,14},
983             {214,12},
984             {257,10},
985             {286,9},
986             {319,8},
987             {360,7},
988             {415,6},
989             {480,5},
990             {562,4},
991             {613,3}
992     };
993
994     float readSharp_cm(unsigned char addr){
995             /* Read value of Sharp sensor mapped on address "addr" and convert to cm
                    distance */
996             float val_cm;
997             unsigned short val_10bit = readSharp(addr);
998             int i;
999             for(i = 0; i<17; i++){
1000                    if (val_10bit >=  sharp_calib[i][0] && val_10bit < sharp_calib[i+1][0]){
1001                            val_cm = (float)sharp_calib[i][1] + (float)((float)val_10bit - (
                                        float) sharp_calib[i][0])* (float)((float)((float)
                                        sharp_calib[i+1][1]- (float)sharp_calib[i][1])/(float)((
                                        float)sharp_calib[i+1][0] - (float)sharp_calib[i][0]));
1002                    return val_cm;
1003                    }
1004            }
1005            return -1; /* Error, data is out of range */
1006    }
1007
1008    float readAccNorm(unsigned char addr){
1009            /* read calibrated and normalized (0-1) value from Accel Channel @ addr */
1010            float norm_val = 0;
```

```
1011            if (addr == 1){
1012                    /* Calibr for X */
1013                    norm_val = (float)((float)readAcc(addr) - (float)325)/(float)164; //325
                            : 0-val , 164 : mod(g)
1014            }
1015            else if (addr == 2){
1016                    /* Calibr for Y */
1017                    norm_val =  (float)((float)readAcc(addr) - (float)338)/(float)162; //339
                            : 0-val , 164 : mod(g)
1018            }
1019            else if (addr == 0){
1020                    /* Calibr for Z */
1021                    norm_val =  (float)((float)readAcc(addr) - (float)286)/(float)164; //286
                            : 0-val , 164 : mod(g)
1022            }
1023            else{
1024                    /* Wrong Channel (addr) selected */
1025                    return -1;
1026            }
1027            /* If you are here then the channel was valid .. */
1028            //Perform value bounding..
1029            if (norm_val > 1)
1030                    norm_val = 1;
1031            else if (norm_val < -1)
1032                    norm_val = -1;
1033            return norm_val;
1034    }
1035
1036    float readAccDeg(unsigned char addr){
1037            /* read deg value from Accel on channel @ addr */
1038            float val = readAccNorm(addr);
1039            //printf("\n\tnormval=%d\n", (int)(1000*val));
1040            val = asin(val);
1041            //printf("\n\tasinval=%d\n", (int)(1000*val));
1042            val = val * 57.296739;//rad2deg
1043            return val;
1044    }
1045
1046    void readAllSensors(Sensors *s){
1047            s->distance_front_center = readSharp_cm(4);
1048            s->distance_floor_center = readSharp_cm(6);
1049            s->distance_floor_left = readSharp_cm(5);
1050            s->distance_floor_right = readSharp_cm(7);
1051            s->body_inclination_xg = readAccDeg(1);
1052            s->body_inclination_yg = readAccDeg(2);
1053            s->body_inclination_zg = readAccDeg(0);
1054            s->abdomen_inclination = getAbdomenDeg();
1055            s->abdomen_inclination_g = getAbdomenDegG();
1056    }
1057
1058    int facingAbyss(void){
1059            //TODO
1060            return -1;
1061    }
1062
1063    //SERIAL void USART_Flush( void ) {
1064            unsigned char dummy;
1065            while ( UCSR1A & (1<<RXC1) )
1066                    dummy = UDR1;
1067    }
```

Appendice B

Manuale Utente

Il manuale utente ha lo scopo di mostrare come interagire con LionHell II e fare da guida nell'installazione dei programmi necessari per la programmazione.

La Sezione B.1 descrive come controllare il robot, come accenderlo e come funziona il telecomando.

La Sezione B.2 fornisce alcune informazioni di carattere generale sulla componentistica di LionHell II e del telecomando.

La Sezione B.3 fornisce gli indirizzi URL utili per il download del software necessario, i manuali utente e le guide, alcuni tutorial e i datasheet dei componenti.

La Sezione B.4 descrive cosa è necessario fare per iniziare a programmare e poter successivamente installare i programmi necessari.

La Sezione B.5 descrive il programma Atmel Studio e i passi necessari per creare un nuovo progetto per poter modificare il firmware di LionHell II.

La Sezione B.6 descrive come trasferire il firmware sulla scheda di controllo CM-510.

La Sezione B.7 mostra come programmare l'XBee e come impostare i valori per poter accedere all'XBee di LionHell II.

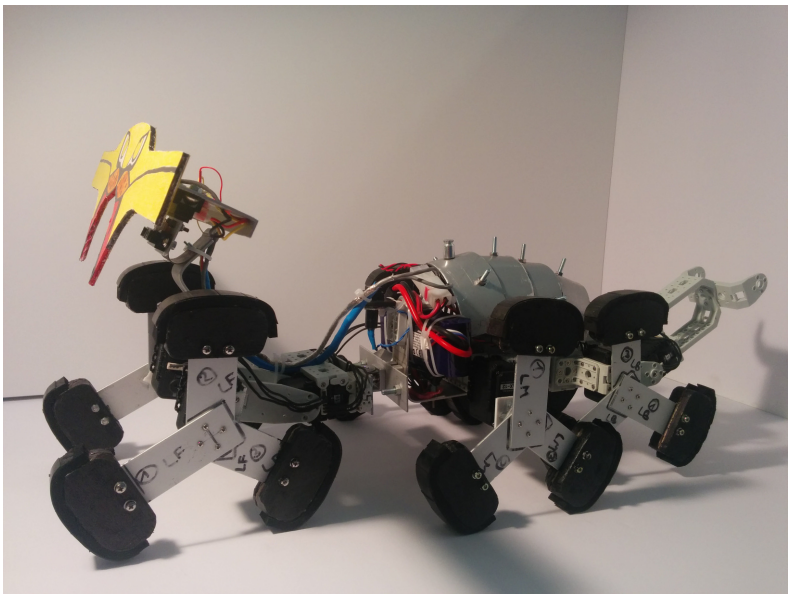
La Sezione B.8, infine, è costituita da una serie di domande e risposte riguardanti LionHell II e il telecomando.

B.1 Come interagire con LionHell II

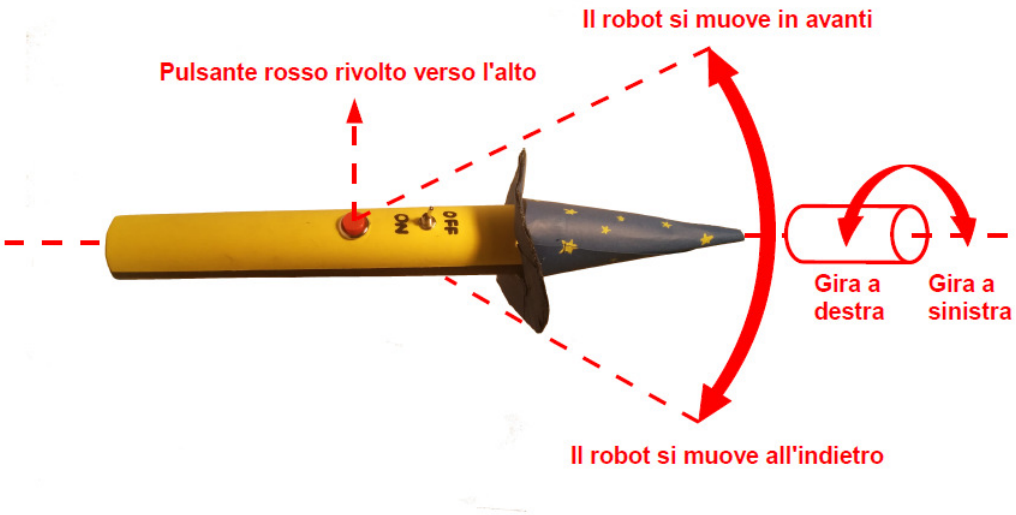
LionHell II è un robot dotato di sei Wheg che gli permettono di scalare piccoli, medi e grandi ostacoli (al massimo pari al doppio della sua altezza), può funzionare per mezzo di una alimentazione a batteria o tramite cavo collegato direttamente alla corrente elettrica. Il controllo di LionHell II avviene per mezzo del telecomando mostrato in Figura B.1b. Il pulsante nero al centro del corpo di LionHell II è il pulsante di accensione, mentre il telecomando è dotato di un interruttore di accensione (ON/OFF) e di un pulsante. Il movimento di LionHell II è possibile solo se si muove il telecomando mentre il pulsante rosso è premuto. Puntando la punta del cappello del telecomando in avanti e con il bottone rosso rivolto verso l'alto è possibile muovere il robot nella direzione desiderata: ruotando il telecomando lungo il proprio asse a destra e a sinistra, LionHell II ruoterà a destra o a sinistra, mentre alzando o abbassando il telecomando è possibile rispettivamente far avanzare o indietreggiare il robot.

Figura B.1: LionHell II e Telecomando

(a) LionHell II



(b) Telecomando



B.2 Informazioni di carattere generale

LionHell II è un robot esapode con le seguenti caratteristiche:

- 10 servomotori attuatori Dynamixel AX-12 appartenenti al kit chiamato Bioloid, commercializzato dalla società Robotis;
- 1 scheda di controllo CM-510, dotata di controller ATMega2561;
- 4 sensori telemetri IR Sharp GP2D120X;
- 1 accelerometro a tre assi MMA7361;
- 1 XBee 4214A;
- 1 XBee Explorer Regulated;
- 1 batteria Nano Tech da 4000 mAh;
- 1 trasformatore AC-DC con tensione di uscita 12 V e corrente massima 5 A;
- 1 pulsante nero stabile normalmente aperto ON/OFF.

Il telecomando è dotato di:

- 1 batteria da 9 V e 250 mAh;
- 1 XBee Explorer Regulated;
- 1 XBee 4214A;
- 1 accelerometro analogico a tre assi ADXL335;
- 1 interruttore switch ON/OFF;
- 1 pulsante rosso instabile normalmente aperto.

B.3 Indirizzi URL utili

Di seguito sono elencati gli indirizzi utili riguardanti prevalentemente tutorial, download di programmi e datasheet dei componenti (ultimo accesso: 2015-03-23):

- http://www.robotgarage.org/wiki/index.php?title=LionHell_McMillan#Download: il sito di LionHell II, con alcuni dati tecnici e da cui è possibile scaricare il firmware del robot;
- http://support.robotis.com/en/: il manuale di supporto della Robotis, particolarmente utile all'inizio;
- http://winavr.sourceforge.net/: il link di download di WinAVR, fondamentale per programmare in AVR;
- http://www.atmel.com/microsite/atmel_studio6/: il link di download di Atmel Studio, necessario per modificare il firmware;
- http://x-ctu.software.informer.com/download/: il link di download di X-CTU, necessario per programmare l'XBee;
- http://roboplus.software.informer.com/download/: il link di download di RoboPlus, necessario per trasferire il firmware modificato su LionHell II;
- http://www.libelium.com/development/waspmote/documentation/x-ctu-tutorial/: Tutorial di X-CTU;
- http://matteo.luccalug.it/wp-content/uploads/2011/11/easy_bee.pdf: Tutorial in italiano dell'XBee;
- http://www.jsjf.demon.co.uk/xbee/xbee.pdf: Guida ufficiale in inglese dell'XBee;
- http://support.robotis.com/en/software/roboplus/roboplus_terminal_main.htm: Guida al terminale RoboPlus;

- http://www.produktinfo.conrad.com/datenblaetter/250000-274999/261767-an-01-ml-Power_Peak_A4_EQ_LCD_230V_12V_de_en_fr_es.pdf: Manuale utente del carica batterie Power Peak A4 EQ-LCD;
- http://www.sharpsma.com/webfm_send/1205: datasheet del telemetro IR Sharp GP2D120X;
- http://www.dfrobot.com/wiki/index.php/SHARP_GP2D120X_IR_ranger_sensor_(4-30cm)_(SKU:SEN0143): link della dfrobot che descrive il codice del telemetro IR GP2D120X;
- http://support.robotis.com/en/product/auxdevice/controller/cm510_manual.htm: datasheet e manuale della scheda di controllo CM-510;
- http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm: datasheet e manuale dell'attuatore AX-12;
- http://www.apexelectrix.com/PDFs/MMA7361/MMA7361_module_datasheet.pdf: datasheet del modulo dell'accelerometro a tre assi MMA7361;
- http://www.apexelectrix.com/PDFs/MMA7361L.pdf: datasheet del chip dell'accelerometro a tre assi MMA7361;
- http://www.hobbyking.com/hobbyking/store/__21939__Turnigy_nano_tech_4000mah_3S_40_80C_Lipo_Pack_TRA2849_Slash_Rustler_Bandit_Stampede_compatible_.htm: dati riguardanti la batteria Nano Tech utilizzata;
- https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf: datasheet dell'XBee 4214A;
- http://www.homotix.it/catalogo/moduli-xbee-e-accessori-689/xbee-explorer-regulated: dati riguardanti l'XBee Explorer Regulated;
- http://www.homotix.it/catalogo/accelerometri/triple-axis-accelerometer-breakout-adxl335: dati riguardanti l'accelerometro del telecomando ADXL335;

- http://www.droids.it/data_sheets/990.006%20datasheet.pdf: datasheet dell'XBee DTE serial board, utilizzata per connettere l'XBee al computer;
- http://support.robotis.com/en/product/auxdevice/interface/ln101_manual.htm: datasheet del cavo di programmazione LN-101.

B.4 Per iniziare

Questa guida si propone di essere un aiuto nei passi necessari riguardanti l'installazione dei programmi per la modifica del firmware di LionHell II e la modifica dei valori dell'XBee. I programmi sono stati testati ed installati su un sistema operativo Windows 7.

Il firmware di LionHell II è salvato all'interno del file "AVRGCC1.c" ed è scritto in codice C-embedded, ma per poterlo modificare e potersi connettere al robot per trasferire il firmware così modificato è necessario seguire alcuni passi fondamentali, descritti alla pagina Robotis e-Manual (Home > Software Help > Software development kit > Embedded C > CM510/CM700 > Getting Started), che corrisponde alla pagina di supporto Robotis e-Manual (si noti che il manuale è scritto in lingua Coreana, Inglese, Giapponese e Cinese):

1. scaricare ed installare WinAVR, necessario per la programmazione AVR, seguendo la procedura indicata nel manuale Robotis e-Manual (Home > Software Help > Software development kit > Embedded C > CM510/CM700 > Getting Started > Installing WinAVR);
2. scaricare ed installare il software Atmel Studio necessario per modificare il firmware come mostrato nel manuale Robotis e-Manual (Home > Software Help > Software development kit > Embedded C > CM510/CM700 > Getting Started > Installing Atmel Studio);
3. preparare l'ambiente di lavoro del software Atmel Studio come mostrato nel manuale Robotis e-Manual (Home > Software Help > Software

development kit > Embedded C > CM510/CM700 > Getting Started > Setting Environment) ignorandone l'esempio;

4. accedere alla pagina di LionHell Mc Millan da cui è possibile scaricare il firmware, o utilizzare il disco su cui è presente il codice necessario (si noti che solo il file AVRGCC1.c è fondamentale ai fini della programmazione del codice di LionHell II);

5. accedere al manuale Robotis e-Manual (Home > Software Help > Software development kit > Embedded C > CM510/CM700) e scaricare il file "embeded_c(cm510_v1.02).zip" e successivamente decomprimerlo.

NOTE: Si noti che il carattere "W barrato" indica il carattere "\" in Coreano.

B.5 Creare un progetto in Atmel Studio

Per creare un nuovo progetto "LionHell" è necessario aprire Atmel Studio precedentemente installato (può metterci parecchio tempo) e seguire la seguente procedura:

1. creare un nuovo progetto "File > New > Project";

2. nella nuova schermata aperta, nella sezione "Name" digitare il nome del progetto "LionHell";

3. selezionare "GCC C Executable Project" in corrispondenza di "Recent-Templates > Installed Templates > C/C++" e premere OK

4. selezionare "ATmega2561" come indicato alla pagina riguardante LionHell Mc Millan (App./Boot Memory (Kbytes) = 256, Data Memory (bytes) = 8192, EEPROM (bytes) = 4096) e premere OK;

5. il progetto è stato creato, ma il file "AVRGCC1.c" non è ancora stato aggiunto: premere con il pulsante destro sulla scritta in grassetto "LionHell" (o il nome del progetto) presente all'interno del riquadro "Solution Explorer" e selezionare "Add > Existing item" e selezionare il file "AVRGCC1.c", quindi premere OK;

6. a questo punto è necessario modificare alcuni valori nelle proprietà del progetto stesso, raggiungibili tramite il percorso “Project > LionHell Properties...”;

7. nella nuova sezione aperta (avrà il nome del progetto, in questo caso “LionHell”) selezionare “Toolchain > AVR/GNU C Compiler > Directories” e in corrispondenza di “Include Paths”, premere sull’icona  , quindi selezionare  e scrivere l’indirizzo della cartella “include” presente all’interno del file “embeded_c(cm510_v1.02).zip” decompresso, con la spunta su “Relative Path” (come nell’esempio “.../embeded_c(cm510_v1.02)/include”), e premere su OK;

8. selezionare “Toolchain > AVR/GNU Linker > Libraries” e nel riquadro basso “Library search path (-WI,L)”, premere sull’icona  , quindi selezionare  e scrivere l’indirizzo della cartella “lib” presente all’interno del file “embeded_c(cm510_v1.02).zip” decompresso precedentemente, senza la spunta su “Relative Path” (come nell’esempio “C:\embeded_c(cm510_v1.02)\lib”), e premere su OK;

9. selezionare “Toolchain > AVR/GNU Linker > Libraries” e nel riquadro alto “Libraries (-WI,-I)” premere sull’icona  e digitare “libdynamixel.a” e premere OK, ripetendo l’operazione anche per le librerie “libserial.a” e “libzigbee.a”;

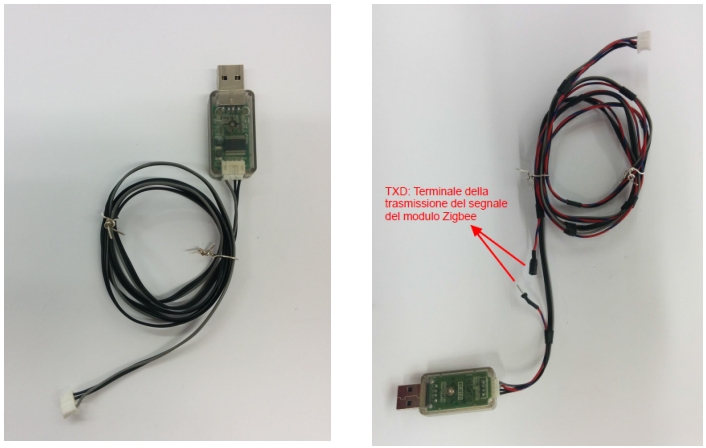
10. salvare le modifiche apportate alle proprietà del progetto;

Per mezzo di Atmel Studio sarà quindi possibile modificare e successivamente aggiornare LionHell II.

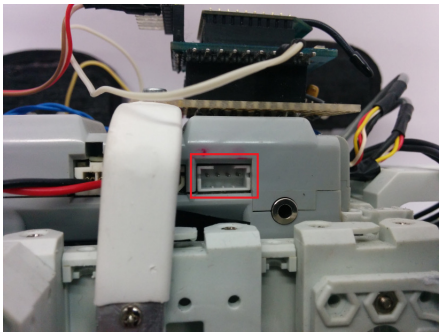
Per ottenere una più chiara lettura del codice, consigliamo di attivare i numeri di fianco alla linea di codice (per fare questo è necessario selezionare Tools > Options > Text Editor > GCC > Display e selezionare “Line numbers”).

Figura B.2: Cavi di programmazione con CM-510 e relativa porta

(a) Cavo di programmazione A (b) Cavo di programmazione B



(c) Porta di programmazione



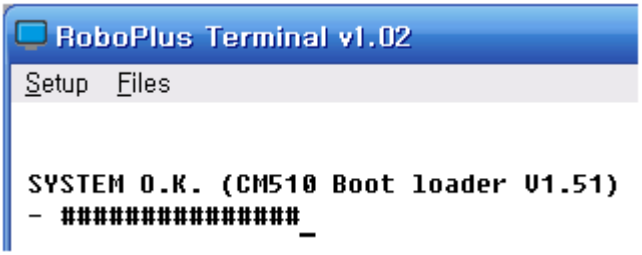
B.6 Trasferire il firmware modificato

Per trasferire il firmware modificato è necessario utilizzare il cavo (LN-101) mostrato in Figura B.2a o quello di Figura B.2b e connetterlo alla porta della scheda di controllo CM-510 mostrata in Figura B.2c, evidenziata da un rettangolo rosso.

Per poter interagire con LionHell II, è necessario scaricare ed installare RoboPlus, ed avviarne il terminale. Una volta avviato il terminale, seguire la seguente procedura:

1. per entrare nel boot loader, tenere premuto il bottone "#" (Shift + 3) mentre si accende il controller o si preme il pulsante si reset posizionato

Figura B.3: Terminale RoboPlus, si è entrati nel boot loader



sulla scheda di controllo. Se l'operazione è stata eseguita correttamente, apparirà la scritta mostrata in Figura B.3;

2. premere su "setup > connect";

3. nella nuova finestra che si aprirà, impostare:

   - Port: COM1
   - Baudrate: 57600 bps

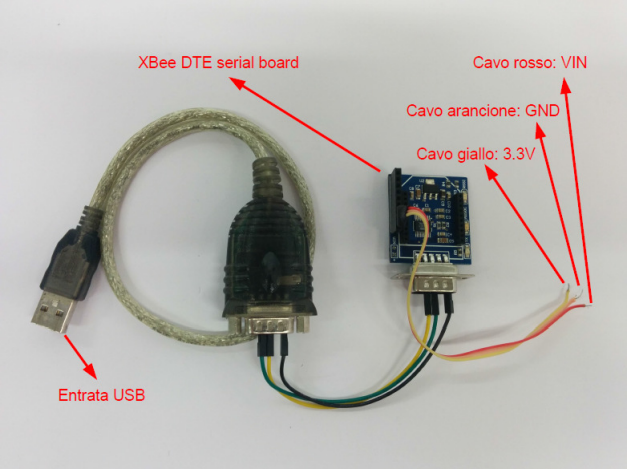
4. ora il PC è connesso con la scheda di controllo CM-510;

5. per trasmettere un file, in questo caso il firmware modificato, premere su "Files > Trasmit file" e caricare il file;

6. per terminare il lavoro, avviare il robot con il cavo di programmazione ancora connesso, digitando "GO" (il programma è eseguito dall'indirizzo 0) o "G [indirizzo]" (il programma è eseguito a partire dall'indirizzo selezionato, scritto in esadecimale) nel terminale (nel caso sia previsto che il robot si muova non appena premiate il pulsante di accensione, state attenti al cavo) oppure resettando la scheda di controllo.

Ulteriori informazioni sono presenti nel manuale Robotis e-Manual (Home > Software Help > Software development kit > Embedded C > CM510/CM700 > Programming > Boot Loader) e alla pagina del terminale RoboPlus.

Figura B.4: Cavo di connessione XBee - PC



B.7 Come programmare l'XBee

Per programmare l'XBee è necessario scaricare ed installare un programma chiamato X-CTU. Per procedere all'installazione e avere un tutorial sul funzionamento del programma, riferirsi al Tutorial - X-CTU.

Una volta installato X-CTU, è necessario collegare l'XBee che si vuole modificare al cavo di connessione mostrato in Figura B.4 (costituito da un XBee DTE serial board), collegare il cavo al computer e procedere all'avvio del programma.

Una volta avviato X-CTU, per modificare e rilevare l'XBee, è necessario:

1. premere sull'icona per cercare la presenza di XBee connessi;
2. selezionare la porta USB corrispondente all'XBee connesso al computer e premere su next;
3. affinché la ricerca vada a buon fine e sia possibile configurare l'XBee, è necessario inserire i seguenti dati:
   - Baud rate: 115200
   - Data bit: 8

- Parity: none
- Stop bits: 1
- Flow control: none

4. a questo punto si è connessi all'XBee ed è possibile effettuare le modifiche del caso.

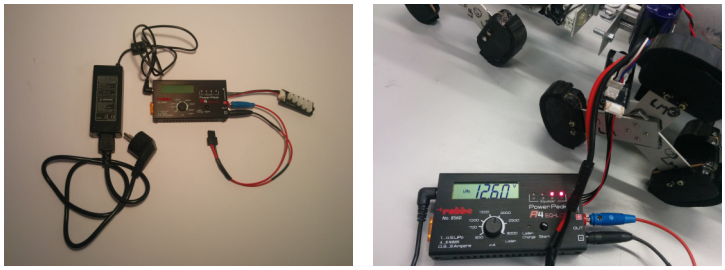
Per ottenere maggiori informazioni riguardanti l'XBee riferirsi al Datasheet -XBee 4214A, al manuale utente Tutorial - XBee in italiano e alla Guida - XBee in inglese, fondamentali per comprenderne il funzionamento.

B.8 Domande e Risposte

- Come faccio a ricaricare la batteria Nano Tech di LionHell II?
  - La batteria è ricaricabile per mezzo del carica batterie Power Peak A4 EQ-LCD mostrato in Figura B.5a (la tensione in ingresso deve essere compresa tra i 12 e i 15 V,consigliabile 14 V). La batteria da ricaricare va connessa al cavo sensore di voltaggio, che misura la tensione, la carica massima e la corrente in ingresso: una volta connesso alla scheda di controllo e ai cavi di carica come mostrato in Figura B.5b, è possibile avviare la ricarica tenendo premuto per circa 10 secondi il pulsante start (è possibile riconoscere la

Figura B.5: Carica batteria per LionHell II

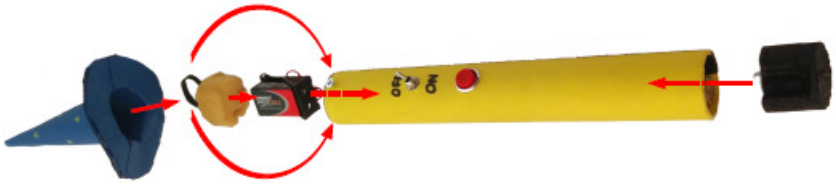
(a) Disconnesso (b) In carica



carica in corso dal led dello stato verde che lampeggia due o tre volte ogni dieci o quindici secondi). Quando la batteria è carica al 95%, il led di stato diventerà arancione, mentre quando la batteria è carica totalmente il carica batteria suona per 15 secondi. Per maggiori informazioni riguardanti il carica batteria ci si riferisca al Manuale utente - Power Peak A4 EQ-LCD;

- Come faccio a sostituire le pile e modificare l'XBee installato nel telecomando?
  - Il telecomando è smontabile come mostrato in Figura B.6 e la batteria di cui ha bisogno è una batteria da 9 V e 250 mAh;
- Il robot si blocca improvvisamente senza motivo, mentre altre volte funziona senza problemi, come mai?
  - Alcuni cavi sono stati fatti a mano e talvolta la connessione potrebbe saltare, controlla le entrate dei cavi nella scheda CM-510 se sono rovinati o se semplicemente non sono bloccati bene;
- I motori fanno rumori strani, come se scattassero, cosa devo fare?
  - L'ingranaggio interno dell'attuatore AX-12 del robot si è danneggiato, è necessario rimuovere l'attuatore e ripararlo, o sostituirlo con uno funzionante. Per smontare ed aprire l'attuatore danneggiato, si faccia riferimento alla guida riguardante l'attuatore AX-12, in fondo alla pagina;
- Alcuni attuatori improvvisamente non si muovono, cosa sta succedendo?
  - Controlla che tutti i cavi degli attuatori siano connessi tra loro e incastrati bene, e nel caso controlla se i cavi degli attuatori si siano danneggiati o meno, dato che raramente si tratta dell'attuatore stesso rotto;

Figura B.6: Telecomando smontato



- I giunti motorizzati centrali del robot muovono la parte frontale del robot in su e in giù ripetutamente, e dopo pochi secondi si blocca tutto, come mai?

  – Vi è un ostacolo che blocca la visuale dei telemetri IR Sharp GP2D120X. Si noti che i sensori individuano la presenza di ostacoli in un raggio di 10°-15° intorno al sensore.

Appendice C

Datasheet

Telemetro

Modello: GP2D120X

Produttore: Sharp Corporation

http://www.sharpsma.com/webfm_send/1205

http://www.dfrobot.com/wiki/index.php/SHARP_GP2D120X_IR_ranger_sensor_(4-30cm)_(SKU:SEN0143)

Scheda di Controllo

Modello: CM-510

Produttore: Robotis

http://support.robotis.com/en/product/auxdevice/controller/cm510_manual.htm

Microcontrollore della Scheda di Controllo

Modello: ATMega2561

Produttore: Atmel

http://www.atmel.com/Images/2549S.pdf

Servomotore

Modello: Dynamixel AX-12

Produttore: Robotis

http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm

Accelerometro di LionHell II

Modello: accelerometro a tre assi MMA7361

Produttore: Apex Electrix

http://www.apexelectrix.com/PDFs/MMA7361/MMA7361_module_datasheet.pdf

http://www.apexelectrix.com/PDFs/MMA7361L.pdf

Batteria di LionHell II

Modello: TRA2849

Produttore: Turnigy

http://www.hobbyking.com/hobbyking/store/__21939__Turnigy_nano_tech_4000mah_3S_40_80C_Lipo_Pack_TRA2849_Slash_Rustler_Bandit_Stampede_compatible_.htm

XBee

Modello: 4214A-XBee

Produttore: Sparkfun Electronics

https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf

XBee Explorer Regulated

Modello: XBee-regulated-v14

Produttore: Sparkfun Electronics

http://www.homotix.it/catalogo/moduli-xbee-e-accessori-689/xbee-explorer-regulated

Accelerometro del Telecomando

Modello: ADXL335

Produttore: Sparkfun Electronics

http://www.homotix.it/catalogo/accelerometri/triple-axis-accelerometer-breakout-adxl335

XBee DTE serial board

Modello: XBee DTE serial board 990.006

Produttore: Droids

http://www.droids.it/data_sheets/990.006%20datasheet.pdf

Cavo di Programmazione

Modello: LN-101

Produttore: Robotis

http://support.robotis.com/en/product/auxdevice/interface/ln101_manual.htm