POLITECNICO DI MILANO

Corso di Laurea Magistrale in Ingegneria Informatica Dipartimento di Elettronica e Informazione

DEVELOPMENT OF A ROUGH TERRAIN WHEG ROBOT WITH MORPHOLOGICAL COMPUTATION

AI & R Lab

Laboratorio di Intelligenza Artificiale e Robotica del Politecnico di Milano

Relatore: Prof. Giuseppina Gini

Tesi di Laurea di:

Vittorio Lumare, matricola 722312

Anno Accademico 2011-2012

Table of Contents

Illustration Index

Index of Tables

Attachments

Preface

Mobile robots on Earth are increasing in number, this is a fact.

Althought many of this robots have been purely developed to foster **academic research**, some of them find a place in **other fields**: see NASA rovers, or the new floor-cleaning robots available in all electronics stores around the world.

This project thesis is dedicated to mobile robotics, and I worked on it thinking that in the **near future** mobile robots will be ever more used in **real applications** over time.

I was been fortunate, in my university studies, because I had the opportunity to watch several robotics courses, thing that is not possible in all universities.

I discovered this world and I love it.

I want to **thank** all the professors and associates who transmitted to me the love for robotics and believed in me, giving me the possibility to do **my best** in this pioneristic field.

I want to thank in particular way : ***Michele Folgheraiter*** and ***Giuseppina Gini***.

Big thanks even to: *Andrea Bonarini* and *Matteo Matteucci*.

Other thanks to *Paolo Belluco* and *Giulio Fontana*, who gave me a big help in preparing me for my first thesis discussion in 2007.

And, obviously, I am grateful to all those people in **AIRLab Lambrate**, who were all working on their projects and made of that Lab an exciting and stimulating place: *Alessandro Nava, Ahmed Ghozia, Davide Medei, Francesco Cartella, Francesco Milli, Maurizio Mercurio, Simone Ceriani* and *Simone Laurenzano*.

I would like to **dedicate** this work to some people **important** to me. Without them I would never had been successful:

my parents: ***Maria De Sena***, ***Antonio Lumare***.

My best teachers: *Maria Voce*, *Giuseppe Sicilia*, *Marilù Cammariere*, *Pippo Olivo, Francesca Maiorano*, *Ferdinando Amendola.*

My friends: *Antonio Maggio, Giovanni Maggio, Domenico Ranieri e Alessandro Sala.*

People who ever believed in me: *Silvano Tricoli*, *Antonio Ursoleo.*

Abstract

This report is a master thesis in robotics, and is about building and testing a mobile robot equipped with a new concept locomotion system: **wheg**.

I've built a **mobile robot**, equipped with **6 whegs** , a **tail** and a **body joint** (which permits more **flexibility** and **adaptation to terrain**) , a **3-axis accelerometer** and a head containing an **infrared sensors array** used to **perceive terrain.**

Several experiments have been made, and a **final structure** has been found, permitting robot to **overcome obstacles** bigger than itself.

The **sensorial perception system** has been tested and modified several times, finally finding a **solution** that allows robot to **perceive terrain**, in order to position itself in the best way to **approach terrain unevenness**.

The final builded robot has been **tested outdoor** on **rough terrain**, with **positive outcome**: the robot has been able to overcome several types of obstacles, even taller than itself.

Finally, three software plugins for the **Player server** (it's a server used for mobile robotics applications) have been implemented, in order to make **simulations** in all supported rendering evironments, like **Stage** and **Gazebo**, or to make **real experiments**, allowing to control the robot with a software **brain** that can be implemented in several programming languages.

Estratto in italiano

In questa tesi di robotica mobile fornirò una descrizione dettagliata del lavoro svolto nel progettare, implementare e testare un nuovo modello di robot mobile per terreni accidentati basato sul mezzo di locomozione Wheg (che sta per wheel + leg).

Il robot costruito è equipaggiato con sei whegs, una coda e un giunto corporeo centrale (che permette più flessibilità e adattamento al terreno), un accelerometro a 3 assi e una testa contenente un'array di sensori di distanza a infrarossi usato per percepire il terreno.

Per mezzo di un simulatore bidimensionale ho creato un modello di robot testandolo e modificandolo iterativamente, fino a giungere ad una versione che permettesse al robot di scavalcare ostacoli più grandi di lui nel più semplice e immediato dei modi.

L'apparato sensoriale che si occupa della percezione del terreno è stato modificato e testato più volte , fino a giungere ad una versione che permettesse al robot di percepire il terreno, con lo scopo di posizionare sé stesso nel migliore dei modi e poter così affrontare le irregolarità del terreno.

Il robot finale costruito è stato testato all'aperto su terreni accidentati, con esito positivo: il robot è riuscito a scavalcare diversi tipi di ostacoli, anche più alti di sé stesso.

Infine, sono stati sviluppati tre plugin software per il server per applicazioni robotiche chiamato ***Player***, in modo da poter accedere alle risorse del robot da remoto e poter effettuare simulazioni negli ambienti supportati, come Stage e Gazebo, o anche esperimenti reali. Il server Player infatti permette di pilotare i robot da remoto (o anche in locale) con un cervello implementato sotto forma di programma, che puo' essere scritto in diversi linguaggi di programmazione.

Sono stati fatti diversi esperimenti indoor e outdoor con il robot costruito. I risultati ottenuti sono positivi per quanto riguarda la struttura ideata e implementata. L'apparato sensoriale si è rivelato insufficiente per una corretta rilevazione delle asperità del terreno.

Lavori futuri possono essere lo sviluppo di un controllo ad alto livello per pianificazione del percorso, lasciando all'attuale firmware il controllo a basso livello dei movimenti e dei comportamenti principali.

Il servomotori usati possono essere sostituiti con modelli economici, per sviluppare una flotta di robot economici e poter così sviluppare applicazioni swarm.

Introduction

If we restrict our view to outdoor mobile robots, we can easily see that the most important aspects are: **locomotion** and **environment perception**.

In this work we focus on both these aspects, concentrating our attention a bit more on the former.

Nature teaches us that locomotion is an aspect intrinsically related to the body.

Body, perception and mind are strictly related, and this concept is well expressed by the term **embodiment**. [1]

This revolutionary concept, mainly used in psychology, has been succesfully applied to autonomous robotics systems, and the key concept is that, in a **physical thinking entity**, the **body** has a strong role in determining the way **environment** is **perceived** by **senses**.

This principle guided me in the **design** of the **Sensorial system**.

The sensorial system functioning has been conceived to be **intrinsic** to the body.

It's **bioinspired**, because it acts in the same way in which animals perceive the terrain and other external entities using **baffles**.

- each sensor is located in the frontal part of the head, and is very near to the perceiving area it should sense, which is in front of the robot, and in this way it interact with the terrain before the rest of the body
- the sensor bar is disposed horizontally with respect to the robot, is large as the robot body, and this is strictly related to the way the robot interact with terrain surface: it needs, in front of itself, an horizontal planar surface parallel to the body orientation, large at least as the body width.

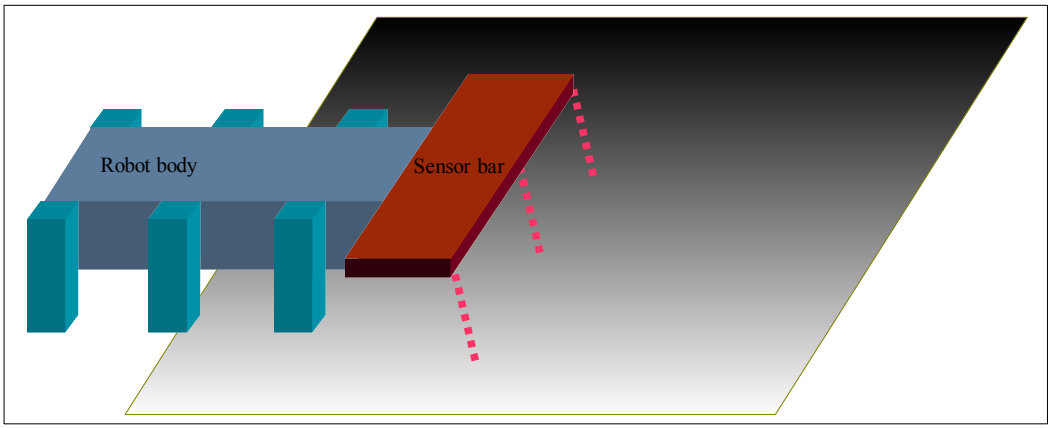


Illustration 1: Bioinspired Perception System

Locomotion system, in an **outdoor** mobile robot, is **critical**, and it must ensure a robust movement control that preserves robot stability.

The faster and most reliable control is given by **morphological computation** [1].

This concept express the fact that the morphology and the materials can take over the functions normally attributed to control.

This concept is perfectly incarnated by the new locomotion system called **wheg** [2][3][4][5][6][7][8][9][10][11].

As we will se in further reading, the morphology of this component release the control from the effort to perform the walking.

Another property of this locomotion system is the easiness of construction and actuation: it has **no moving parts** and a **single motor** is sufficient to drive it.

The **objective** of this work is to **discover** and **test** a **structural design** that **simplifies** the **control** of a **wheg** robot on **rough terrains**.

Consequent purpose of this work is to better understand the **potentiality** and **usefulness** of the **wheg locomotion system**.

Since the **agility** on **rough terrains** would have been a key feature of the robot I was going to build, I needed to **design** the whole robot focusing on this aspect.

In order to **adapt** the robot to **uneven** terrain **surfaces**, its structure should have had some **flexibility**. This has been obtained putting a **joint** in the **center** of robot, allowing it to **tilt** up or down. Adaptation to environment using a **body joint** has been addressed in other recent works [9] [12].

In rough terrain robots terrain adaptation is an essential aspect, and it has been addressed in several studies. In recent works on this topic, **track** was the most used locomotion system [13][14].

The task of **climbing** a **small obstacle** has been performed quite easily with this structure, but robot **stability improvement** made necessary in order to climb **bigger obstacles.**

Robot **stability** has been improved adding a **tail** to the robot.

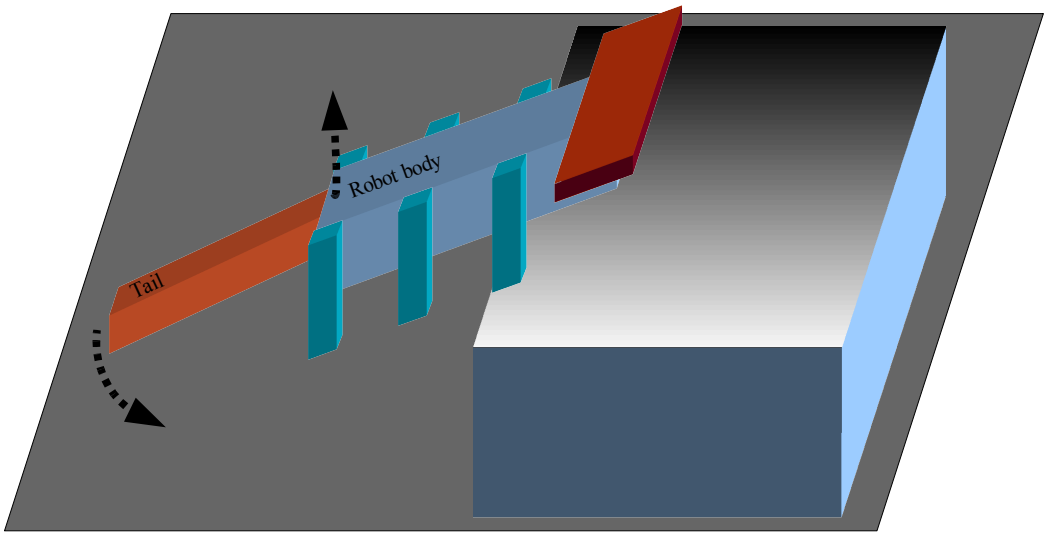


Illustration 2: Climbing with Tail - Part 1

When tail lowers down, robot body lifts up consequently.

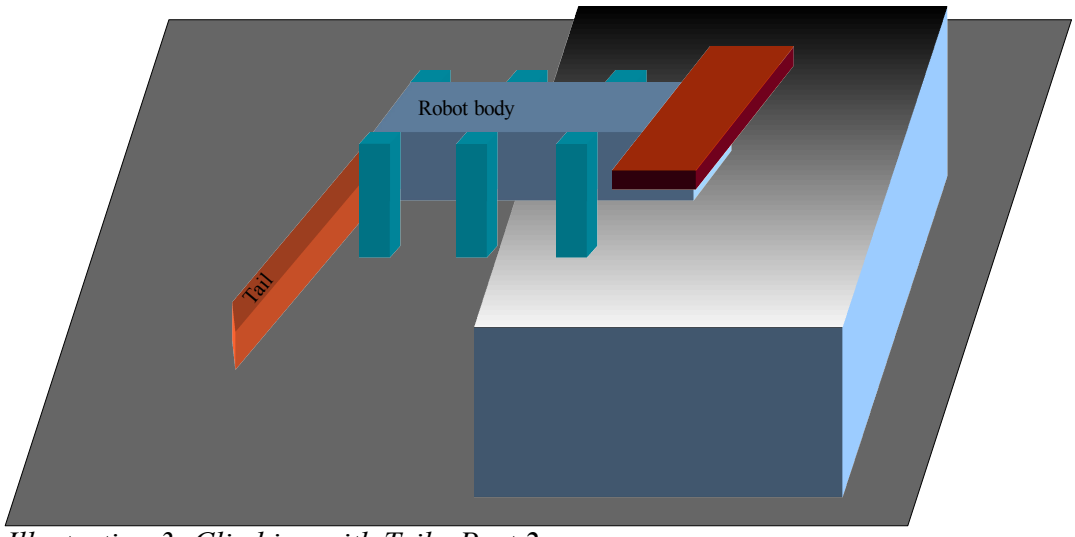


Illustration 3: Climbing with Tail - Part 2

When robot tries to overcome a big obstacle, lifting the frontal part of the body, it starts **falling** backwards due to the **gravity** action: here the tail comes into game. Lowering the tail until it reaches the terrain, the robot can find a **support** to prevent falling backwards.

Due to the big result obtained by this very simple reaction, this can be considered another case of morphological computation.[1]

The involved morphological features are :

- tail and body connection through rotation joint
- length of tail proportioned to body length
- rotational movement of tail with respect to the body

Another effect of this morphological computation is to increase the **grip** on terrain of the frontal whegs, allowing the robot to pull its body up and over the obstacle.

The whegs , and all the other moving parts of robot, have been drived by **servo motors**.

The used servos were used in two different ways: **continuous rotation** mode (for the six whegs) or **position mode** (for the body joint and the tail).

Another stability issue arise when robot is **not balanced** : if the left part of body is at different height with respect to the right part, the robot will **fall laterally**.

Since robot has **no lateral flexibility** (it can only flect his body up and down along his "spine") the best way to face this problem is **prevention**: the robot must be able to **detect** the **terrain** in front of itself, and then **decide** what to do.

If the robot detect an (almost) perfectly **horizontal** terrain **edge** in front of itself, it tries to walk on it, otherwise it **turn** left or right in order to **find** a **more suitable** terrain **configuration**.

Terrain detection is performed through a **sensor array**.

The sensors used are **infrared rangefinders**.

In order to better **react** to the current situation a **3-axis accelerometer** has been mounted in the **center** of robot. It is primarily used to roughly detect the robot **orientation** with respect to the **gravity** force **vector**.

The central **processing unit** of robot is a **microcontroller**, hosted by a **board**. The board is equipped with several **connectors**, allowing a **limited** number of connections, so some **multiplexing** has been necessary to connect all the **peripheral** sensors.

In the first chapter of this thesis, *"State of the Art"* , I'll start describing the **state of the art** of **wheg robots**. I have **classified** wheg robots depending on **which kind of wheg they use**, starting from the most widely used type.

In the chapter entitled *"Simulations and Model Design"* I'll show you the **method** I used to **design** the robot structure. All my **decisions** about design are based on

simulation phases, in which I test the robot model with the help of a bidimesional simulation tool. After each simulation I **analyze** the **behavior** and the **limits** of the **model**, **improving** it and then **testing** the new model in the next simulation. This **simulation / model-improvement sequence** is repeated **iteratively** until the **robot model** become **succesful** in his **task**.

In the following chapter *"Real Robot Implementation"*, as the name suggests, I'll show you the **implementation** of the **real robot** . This is an **adaptation** of the best robot model I've found in the **design** phase, made necessary due to the fact that I couldn't build all the needed parts as I wanted. You'll fine here **photos** of the **two version** of the **robot**, in fact the first version has been soon modified in order to accept a different wheg type.

In the following chapter "*Hardware"* you will find all information about the **electronic** and **electromechanical components** used to build the robot.

In this section are described :

- sensors
- servomotors
- control board
- powering system

For each component I'll list the main technical **specifications**, and **the way I used it**.

In the following chapter *"Firmware"* I'll describe the **main control loop** of the **embedded code** running on the **control board**. Control loop is divided in sections, each of which acts like a **behavior**. Each behavior will be explained, both with words both showing you the implemented code.

Then, in the *"Software"* chapter I'll talk about the **plugins** developed for the **Player server** [15][16], showing you the information each plugin manages. Player is a server for robot interfacing and simulation, and has been chosen because of several reasons: it's open source, it's quite reliable and in constant upgrade.

I'll show you the **output data** each plugin produces when connecting to it, using the client tool *playerprint*.

The next chapter is called *"Experiments"* and I think it's the most exciting, because you will find there all the photos of the **robot in action**. I made some **experiments on the field**, trying to discover the **strength** and the **limits** of the builded robot system. The environment chosen is the more **rough** and **wild** I could find in Milano's parks. I searched for natural terrain surfaces with strong unevenness and natural obstacles .

Finally I used as test fields some structured places containing interesting obstacles like stairs.

In the final chapter , *"Conclusion and Future Work"* I'll comment the **results** obtained, describing the **limits** of the **current** system. Then I'll try to give you some ideas for eventual **future work** on this project.

Chapter 1 State of the Art

Wheg is an acronym standing for **Wheel** + **Leg** .

It's a kind of wheel that **emulates** the **legs** behavior.

A wheg is composed by a **central rotating element**, to which one or more **bars** are fixed. This bars works as legs.

1.1 Three Legs wheg

The typical wheg is composed by three bars, displaced at 120° one from the other, around the central body.

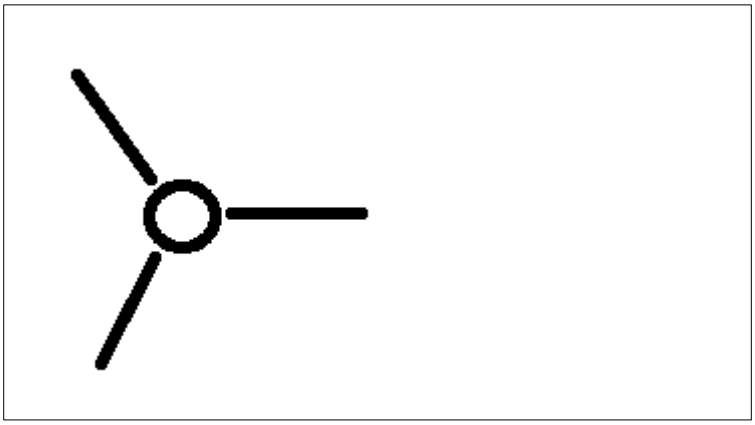


Illustration 4: Three Legs Wheg

Robots equipped with this wheg:

Whegs I [2]

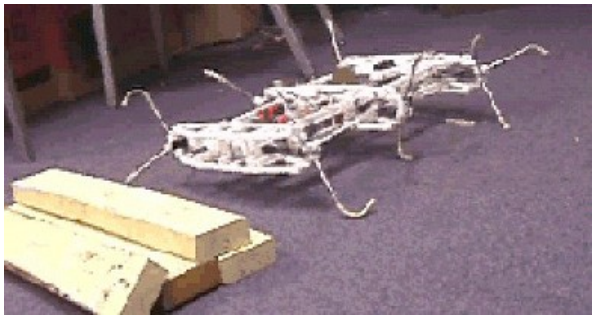


Illustration 5: Whegs I

Whegs II

Illustration 6: Whegs II

Autonomous Whegs II [3][4]

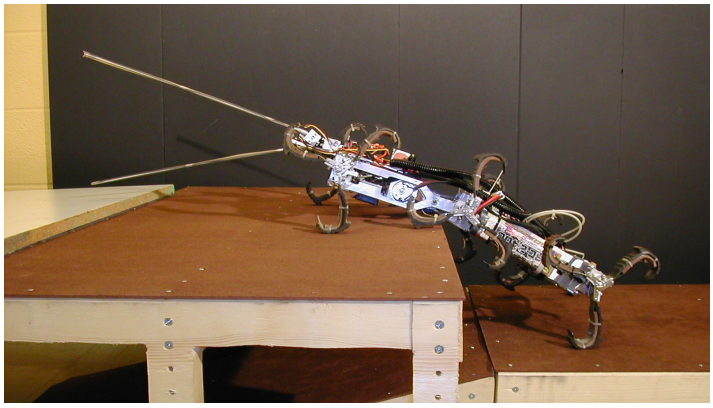


Illustration 7: Autonomous Whegs

Mini Whegs I [5]

Illustration 8: Mini Whegs I

DAGSI Whegs

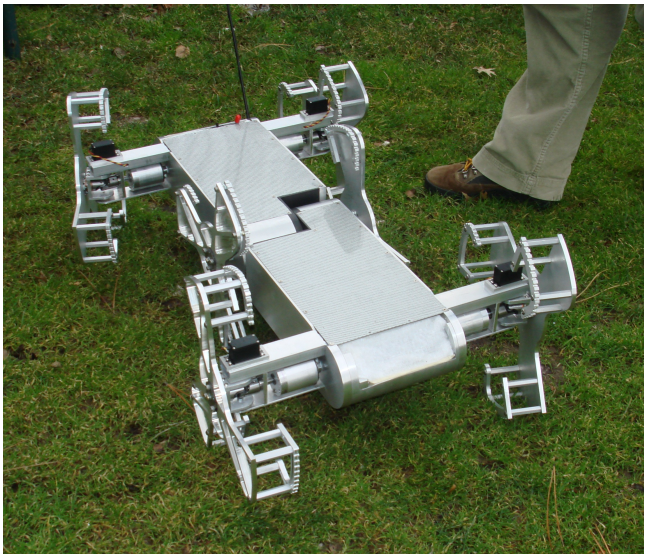


Illustration 9: DAGSI Whegs

Climbing Mini-Whegs™ [6][7][8][9]

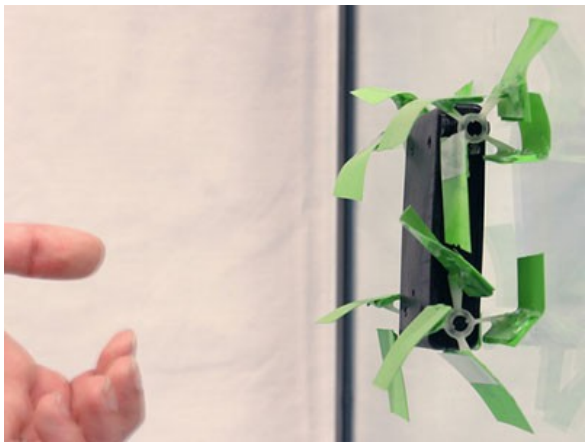


Illustration 10: Climbing Mini-Whegs

1.2 Single Leg wheg

Wheg comes even in a simpler fashion: one leg only.

Illustration 11: Single Leg wheg

The typical robot that uses this kind of whegs has three whegs each side , for a total of six whegs.

The single leg wheg is rapidly actuated, in order to ensure that its end ever touches the floor, giving stable sustain to the robot.

This is the first kind of wheg ever built.

Robots equipped with this wheg:

PROLERO (1996) [10]

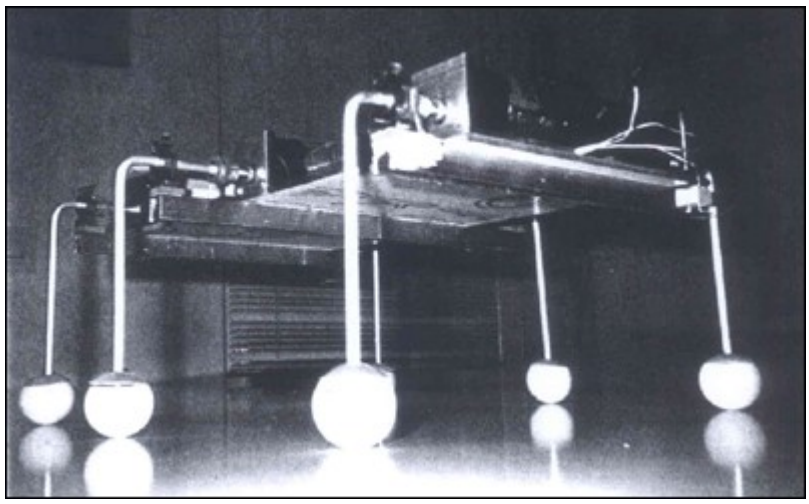


Illustration 12: Prolero

1.3 Curved Leg wheg

This kind of wheg has a single **curved** leg.

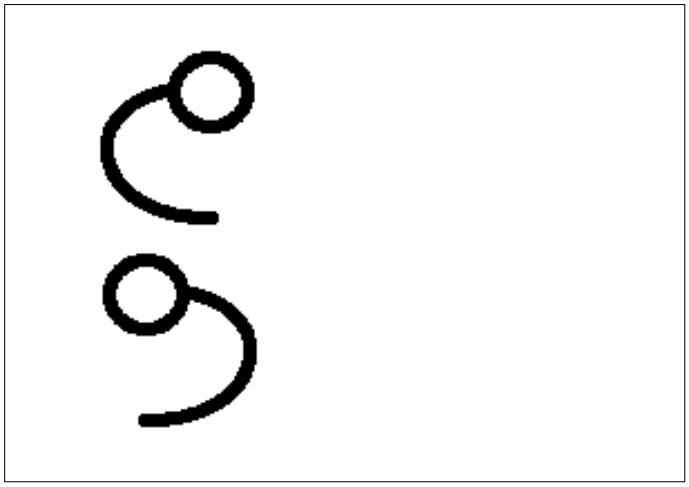


Illustration 13: Curved Leg wheg

It ensures a better grip to terrain.

Same as in single leg wheg, it's actuated with fast movements, and we can find it in robots in a six wheg configuration (three whegs for each side of the robot).

In some robots, leg concavity is oriented in the frontal direction of robot movement, in others in the opposite direction, as we can see in the figure below.

Robots equipped with this wheg:

- Rhex (concavity towards forward movement direction) [2]

Illustration 14: Rhex

1.4 Star wheg

This is the most **complex** wheg ever built.

It's a wheg that can change its shape: when it is closed it is like a standard **circular** wheel, when it's opened it has a **star** shape. Legs are slightly curved in order to ensure a better **grip** .

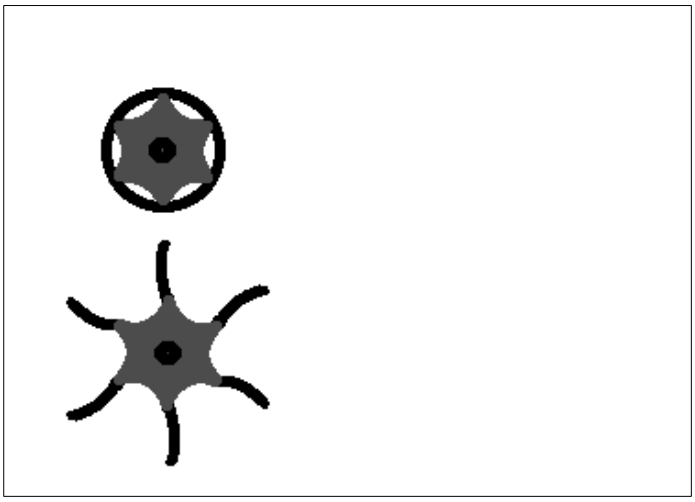


Illustration 15: Star wheg

This wheg has been conceived principally to **climb stairs** (if mounted on an appropriately designed robot).

The star wheg allows to reach **fast speeds** on plain surface terrains (circular configuration) and a good behavior on **rough** terrains (star configuration).

Robots equipped with this wheg: Ratasjalg [11]

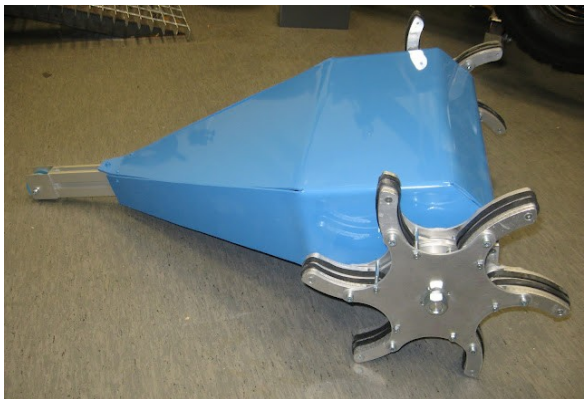


Illustration 16: Ratasjalg

Chapter 2 Simulations and Model Design

Before building the robot, I decided to make some **simulations**, in order to anticipately detect potential **design errors**.

The starting design has been taken from a previous wheg robot prototype: **EMBOT**.

EMBOT stands for **Electro-pneumatic Mobile roBOT**.

Illustration 17: EMBOT

This robot was conceived in order to improve the wheg locomotion system with **electro-pneumatically extensible bars**. The used wheg were six three-legs-whegs, three for each robot side.

The following simulations have been made with a shareware version of *Working Model 2D*. The **simulated** robot has been designed within the simulation tool, respecting **real mass values** and **dimensions** of the **real** robot.

Since the aim is to test a wheg-robot in a **rough** terrain environment, in order to understand the robot's limit of movement , has been designed an **hard** obstacle: a **stair**.

The difficulty of climbing over such an obstacle is due to the **long sequence of single steps**, each of which is not so hard to be climbed, but the **whole** sequence is really problematic for the robot.

The **purpose** was not to build a stair-climbing robot but, designing a robot capable of climbing over a stair, the **final real robot** surely it would have been **able** to move flawlessly on a **very rough terrain**.

Another recent work on the topic of climbing over obstacles in mobile robotics has been made by Hyungseok Kim, Taehun Kang, Vo Gia Loc and Hyouk Ryeol Choi [17].

2.1 First simulation

This is the first simulation and, as we can see from the images, the robot is **too short** to climb over **obstacles** even **smaller than itself**.

At the beginning Embot climbs over the first step succesfully.

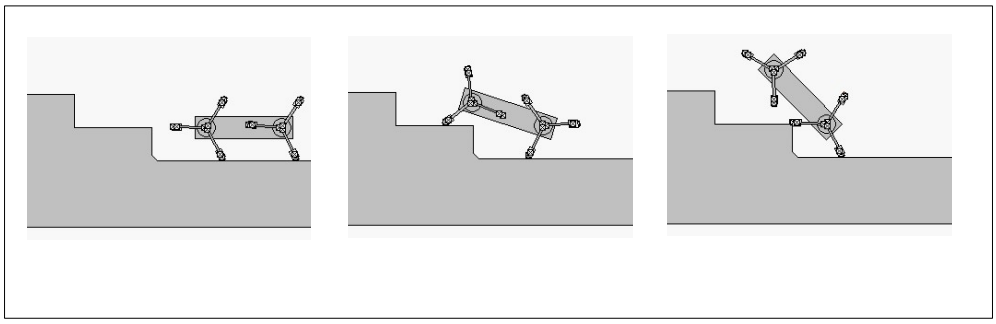


Illustration 18: First Simulation - Step 1

Even here the robo is able to go on.

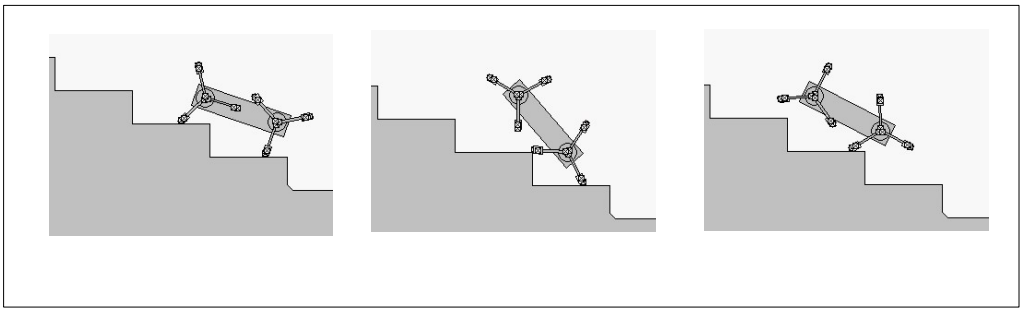


Illustration 19: First Simulation - Step 2

Here its **center of mass** has been moved too much to the **right**: Embot starts **falling backward**.

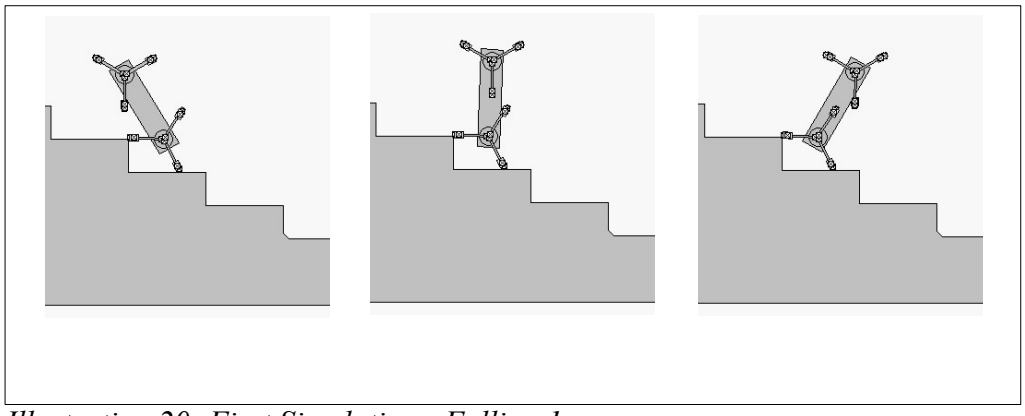


Illustration 20: First Simulation - Falling 1

Due to the high motum quantity reached, Embot is **unable** to **invert** the march **direction**, and continue falling down the stairs..

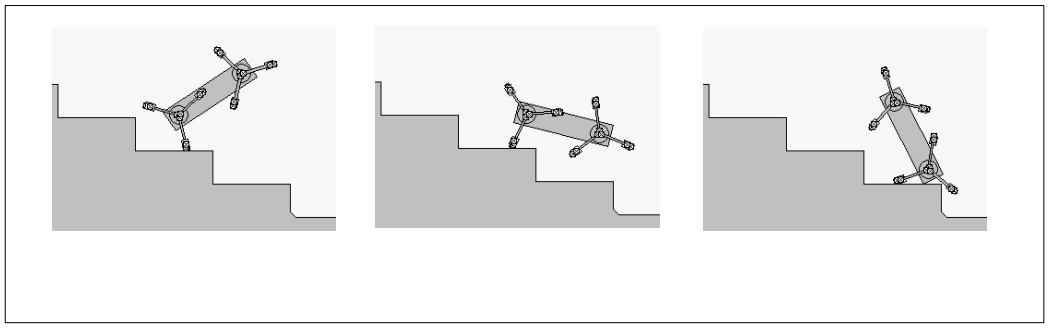


Illustration 21: First Simulation - Falling 2

..until it reaches the ground.

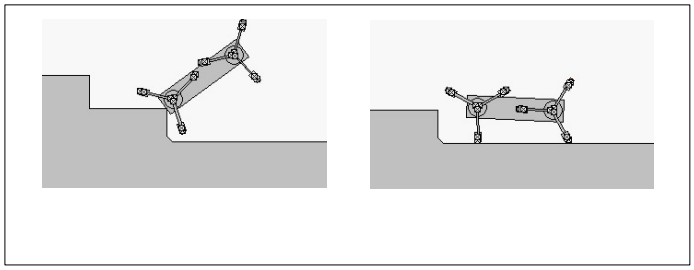


Illustration 22: First Simulation - Falling 3

We can easily understand the reason of this failure: Embot is **too short** to overcome obstacles that are smaller than itself.

I modified its model **increasing** the robot **body length** of about five centimeters.

2.2 Longer body, better results

First step succesfully climbed over.

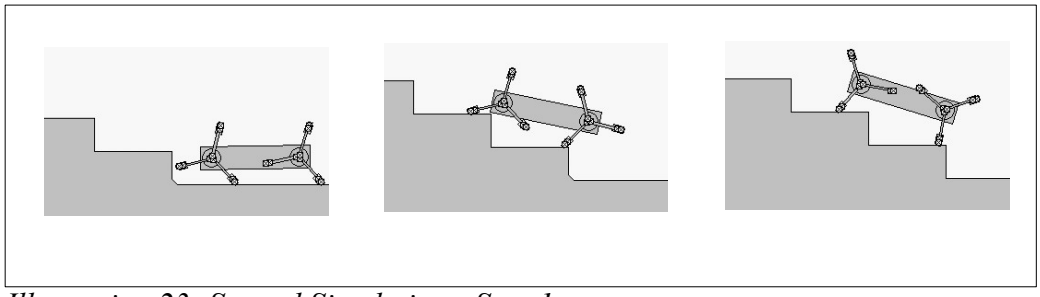


Illustration 23: Second Simulation - Step 1

Embot easily reaches the last step.

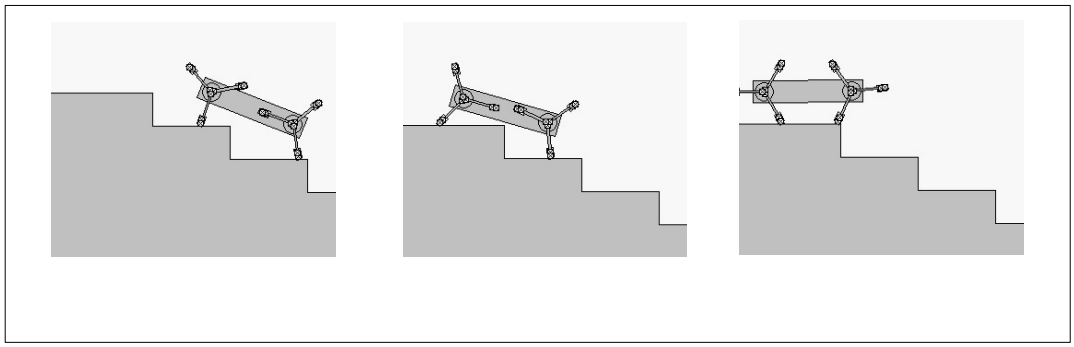


Illustration 24: Second Simulation – Last Step

Simply **extending** of a very **small** quantity the robot body has made possible for Embot to overcome flawlessly each single step of the stairs.

But, as we know, steps are **smaller** than the robot, and we want it to be able to climb over obstacles **bigger** than itself.

In the next simulation we will see what happens with bigger obstacles.

2.3 Obstacles big like robot

Embot starts climbing over the first step...

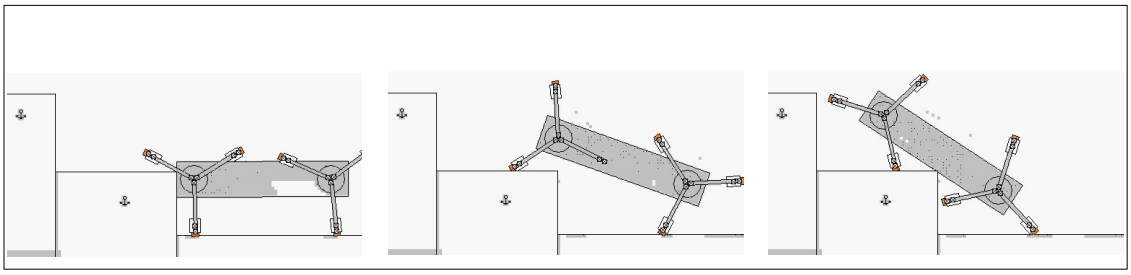


Illustration 25: Third Simulation - Step 1

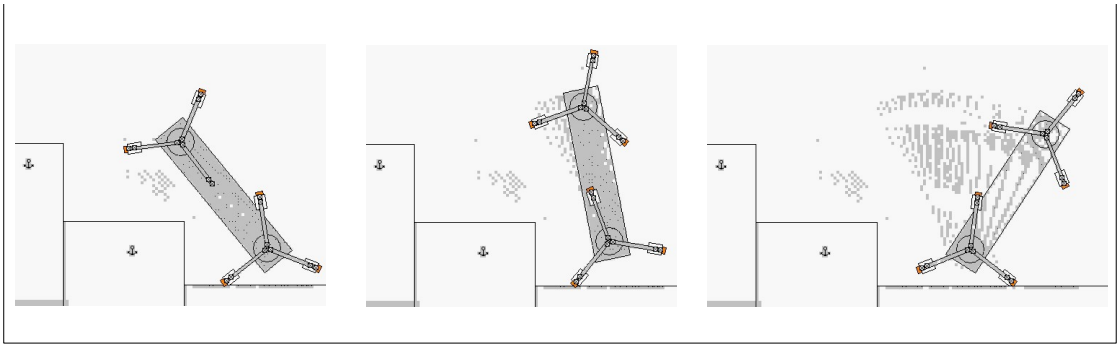


Illustration 26: Third Simulation - Falling

Despite its longer body, the robot is **unable** to climb over the step.

This happens because it has **no support in the back**: when the back wheg (we see only one back wheg because of the 2D simulation) rotates, the mechanical **support** of the **servomotor** is the robot **chassis**, which starts rotating **clockwise**, making the robot **falling** backwards. The solution to this problem is to **avoid** the chassis **rotation**.

How to avoid the chassis rotation? A simple approach is to add a fixed **link** in the **back** , such a kind of **tail**, so that when the chassis starts rotating clockwise , the tail will touch the **ground** surface **blocking** the rotation.

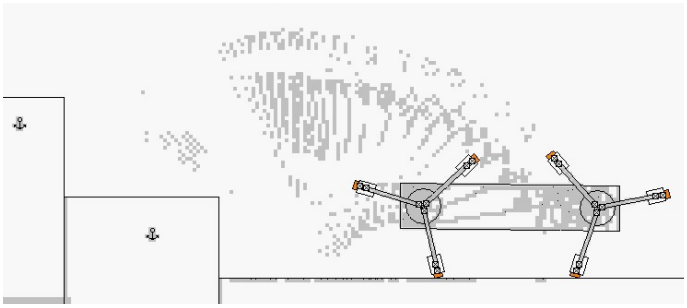


Illustration 27: Third Simulation - Robot Failed

The next robot **design** was conceived thinking to all this issues we have seen in these simulations.

In order to give more **grip** on the terrain , **another wheg** has been added to the bidimensional model, so that the real wheg would have had a total of six whegs.

In the next simulation we'll see this new concept model at work.

2.4 New model

Model **changes**:

- New **central** wheg
- New **link** added to body
- **Joint** in the body center
- **Tail** added to the back

- Wheg **foot design** improved to give better **grip**

Note: in this setting the **tail** has been fixed to the last servomotor, so that when it starts rotating, all the rotating force is **transmitted** to the tail, which consequently transimits it to the **ground**, giving **stable support** to the robot .

We can see the forces vectors in the simulation (red arrows)

The stair-climbing task simulation has been abandoned in favor of a more real-like simulation. I modeled a very rough surface , full of high spikes and pits.

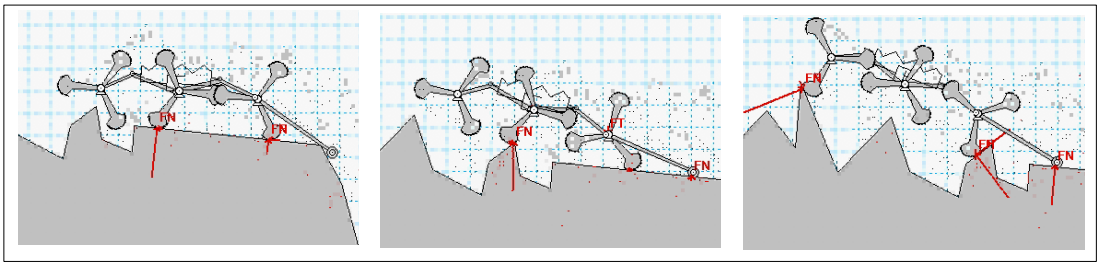


Illustration 28: 4-th Simulation - Start

As we can see , the robot is able to walk on very rough surface terrain.

He goes on even in presence of high surface spikes, thing that would have been not possible for the previous robot model. This **better behavior** is principally due to the new **central wheg** added to the model, and to the slight **body flexibility**, due to the action of a **spring** and a **damper** in the body **center**.

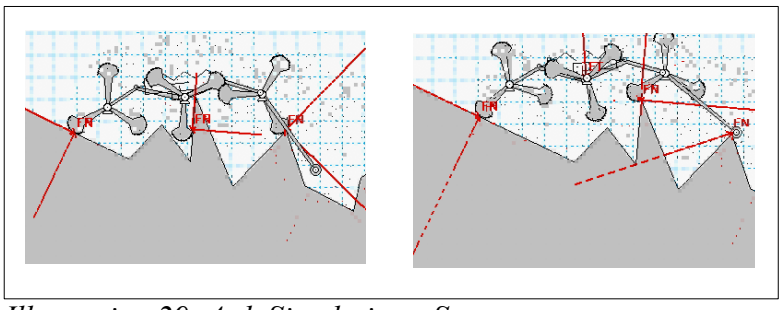


Illustration 29: 4-th Simulation - Success

Thanks to the **tail** action, robot never falls on his back.

Chapter 3 Real Robot Implementation

The first **real robot design** has been **based** on the model design obtained from the results of the previous **simulations**.

The key features are:

- tail
- 3 whegs each side
- central body joint

Since the Embot structure was too **different** from this new model, I and the prof. Gini thinked to build a **new** robot from scratch.

Using some plastic parts contained in Bioloid Basic Kit (produced by Robotis) I've built all the structure. This parts integrate perfectly with the Dynamixel servos.

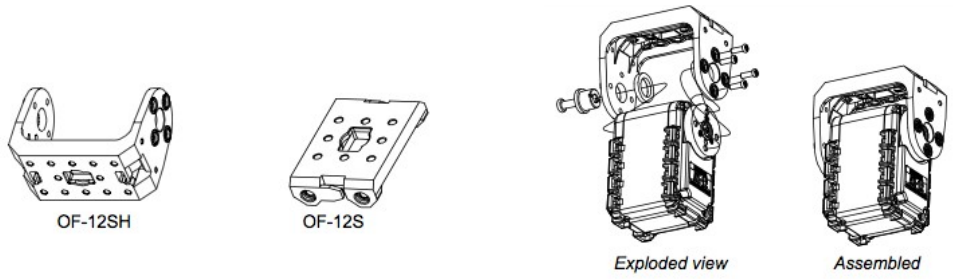


Illustration 30: Structural parts from BIOLOID Robotis KIT

On the first attempt, I ended up with this:

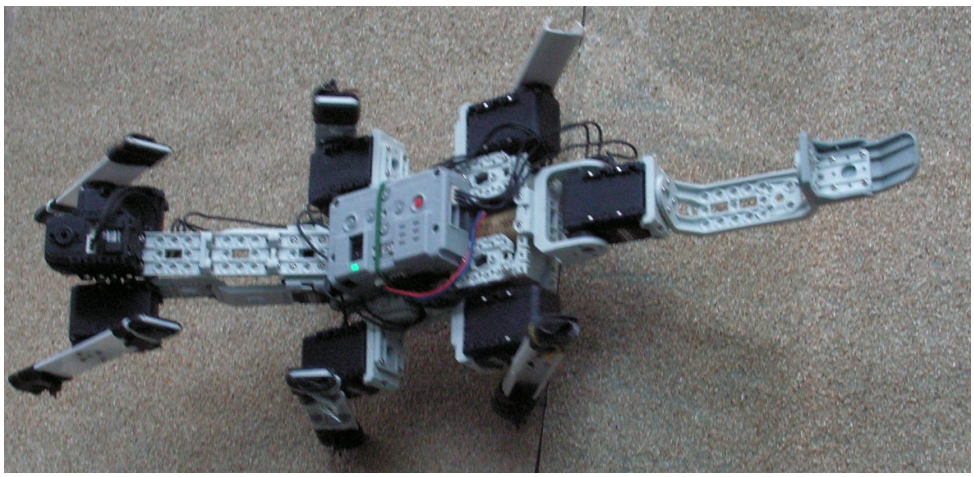


Illustration 31: First Robot Implementation – Photo 1

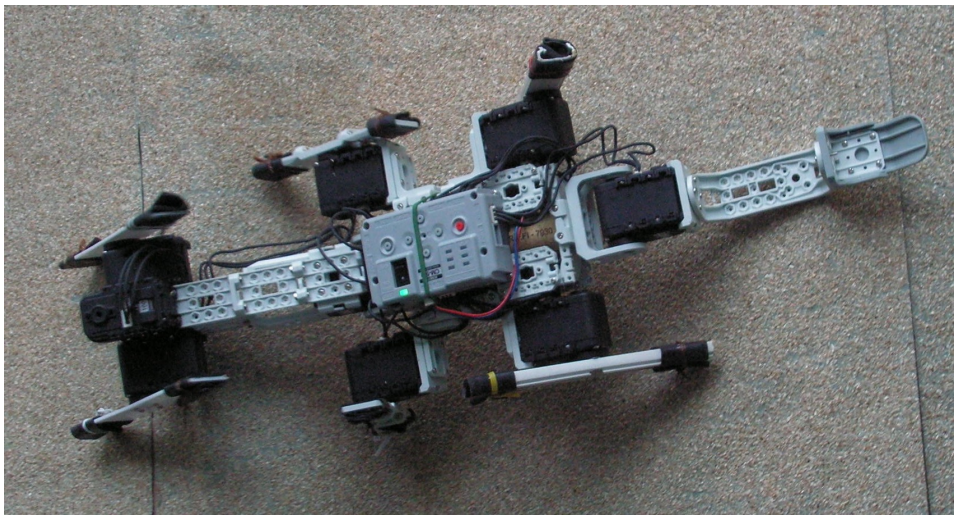


Illustration 32: First Robot Implementation - Photo 2

As we can see, the robot had been equipped with a new type of wheg.

We call it: **Two Legs Wheg.**

At the end of each feet has been fixed a piece of **rubber**, in order to obtain more **grip** on terrain.

This robot design had some **problems**:

- The **two-leg-whegs** caused the robot to **fall** during the climbing phase.
- Whegs material was **plastic**: too easily breakable.
- Legs where too **short**

So the next step of my work was to **change** the whegs with bigger and stronger ones, and of another type.

I ended up with classic **three-legs-whegs**.

I thinked to equip this wheg with the best **feet** I could build, in order to give more **grip** on terrain:

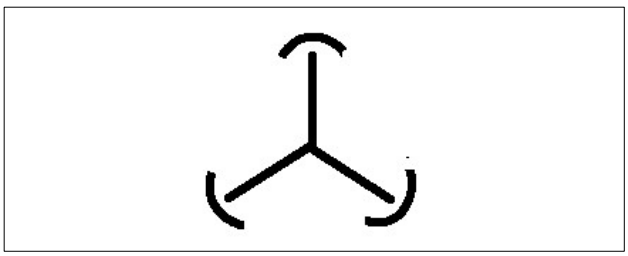


Illustration 33: New Wheg Model with New Feet

This wheg was conceived to be cutted from a single plexiglass sheet.

The complete design is shown in the next figure:

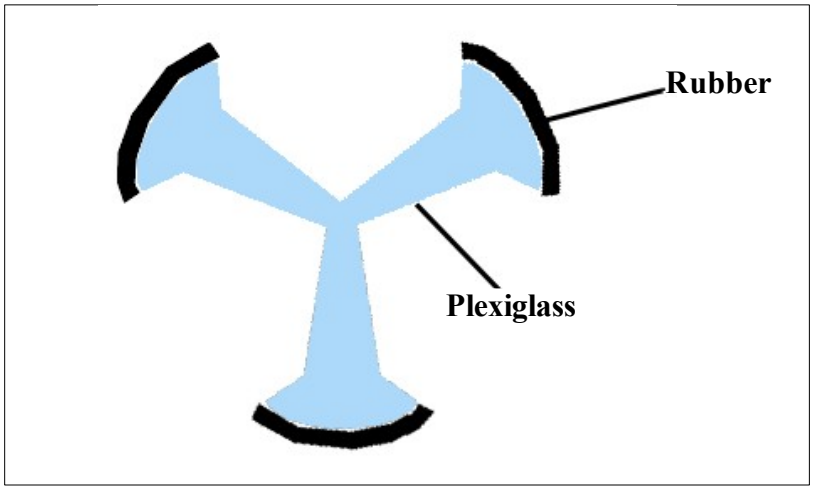


Illustration 34: New Wheg - Complete Design

Unfortunately, **plexiglass** was too **weak** for the purpose, so I changed material again. I chosen **aluminium**, because **stronger** and **lighter**.

Since I could not find any aluminium sheet at affordable price, I decided to abandon the idea of cutting the wheg from a single sheet.

I found some aluminum **bars** in a brico-center, and then assembled them with nuts and bolts:

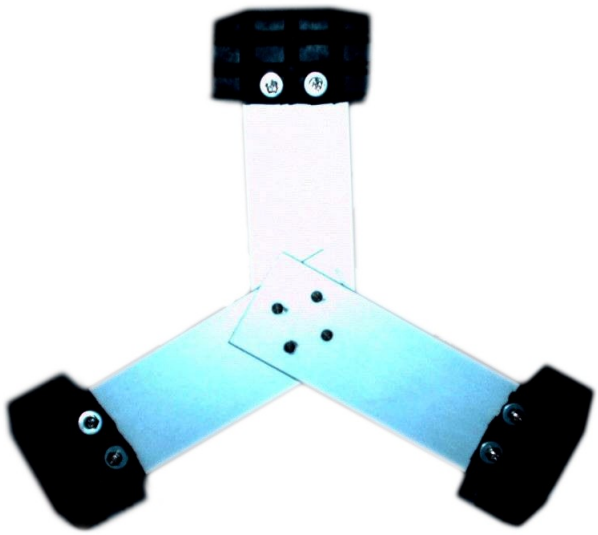


Illustration 35: Builded Wheg

At the end of each bar, I fixed a car-carpet rubber to be used as a feet.

Due to the bigger dimension of this wheg, the **whole robot** had been changed in **size**.

Whegs have been spaced apart from each other horizontally , in order to prevent self collisions during wheg rotation.

The robot's has been **increased** in size to permit a stabler **equilibrium** (robot grew in height, and this caused an elevation of the robot's center of mass).

I've **modeled** some aluminium bars through **flame shaping**, and then I added them to the robot frame.

One bar has been putted between the **central par**t of structure and the **frontal whegs**.

Another aluminium bar has been putted between the sensor-bar and the frontal whegs. The last bar assolves the same **function** of the **neck** of humans and animals.

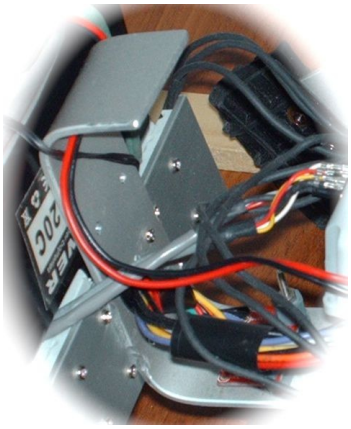


Illustration 36: Aluminum support bar

Illustration 37: Aluminium Neck

Another element to increase in size has been the **tail**: the bigger size of body structure imposed the adoption of a **longer** tail in order to prevent the robot **falling backward**.

The final robot, I called ***LionHell McMillan***, is shown in the following photos:

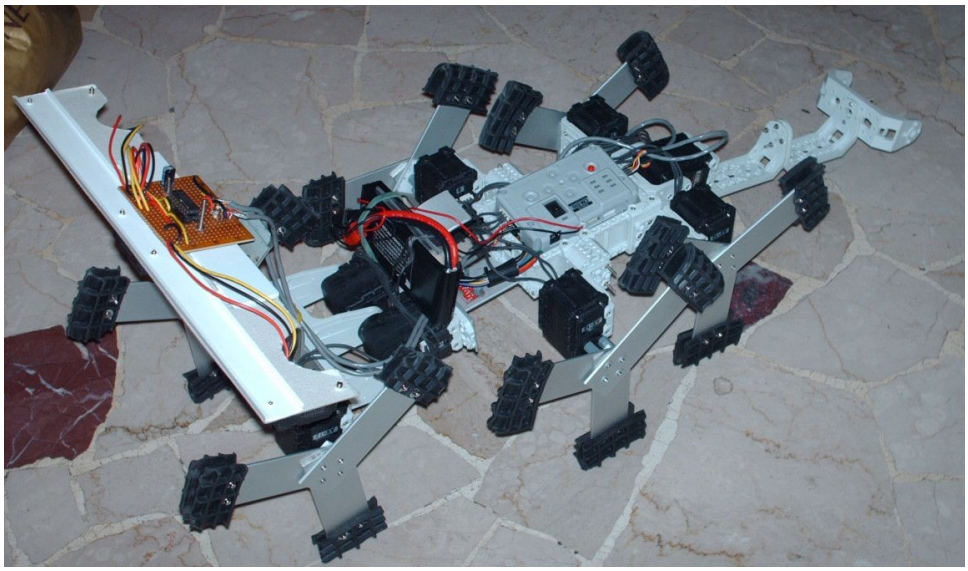


Illustration 38: Final Robot - LionHell McMillan - Photo 1

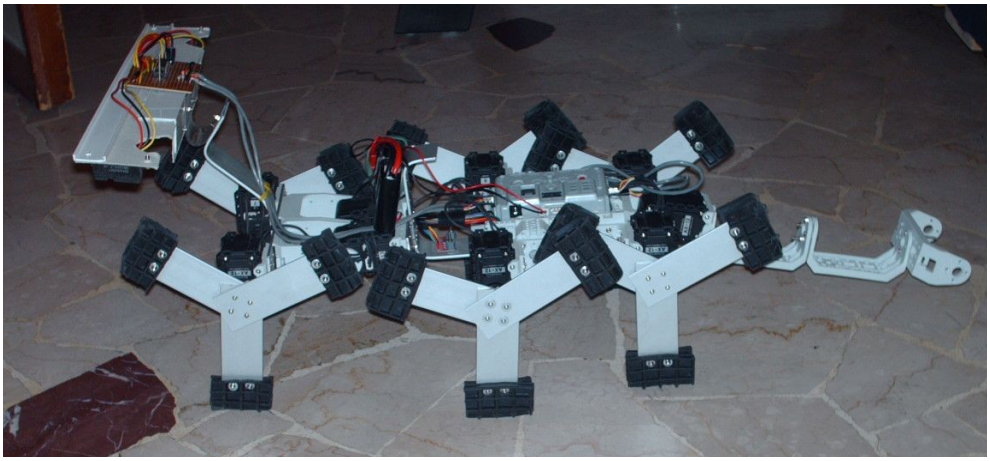


Illustration 39: Final Robot - LionHell McMillan - Photo 2

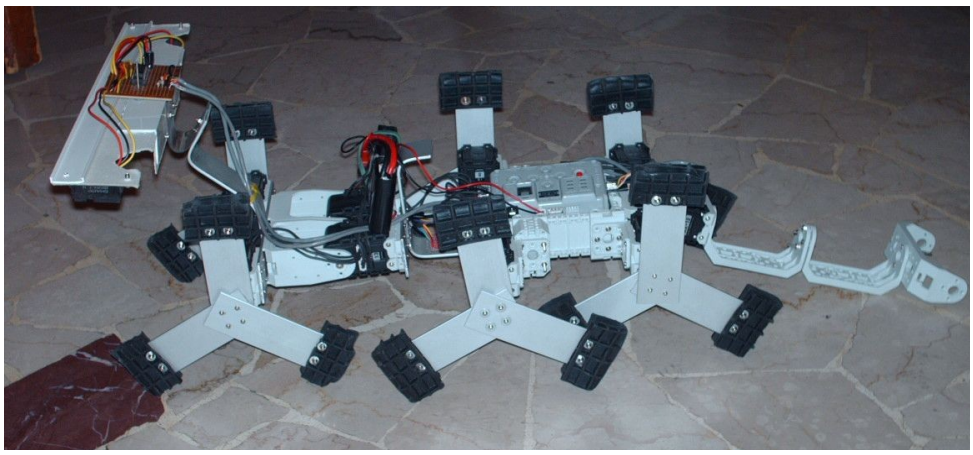


Illustration 40: Final Robot - LionHell McMillan - Photo 3

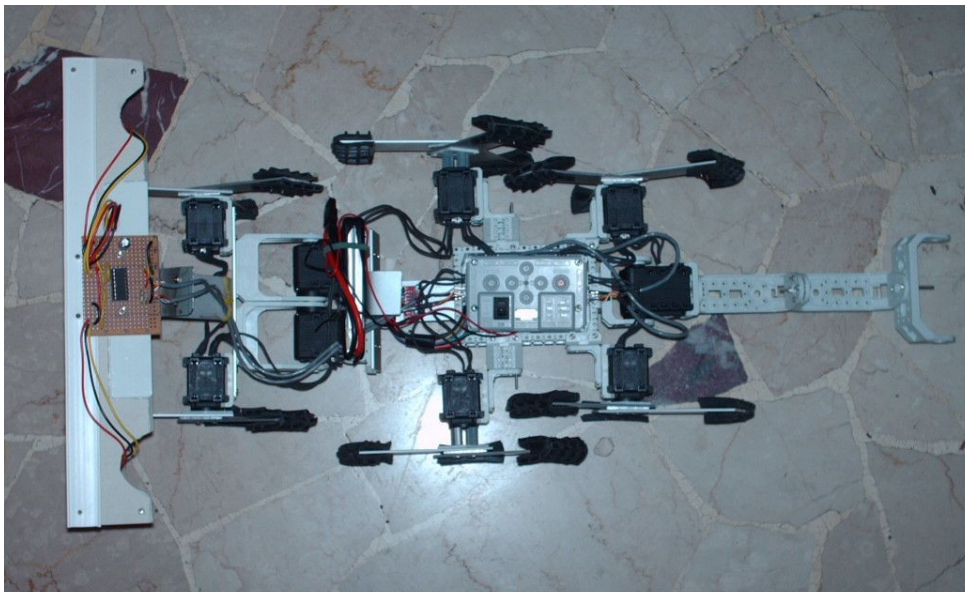


Illustration 41: Final Robot - LionHell McMillan - Photo 4

Chapter 4 Hardware

In this section all the hardware components will be illustrated, excluding structure because it was yet discussed in the previous chapters.

4.1 Servomotors

In this robot the servomotors are the only **actuators.**

These servos have been taken from a Robotics kit called **Bioloid**, commercialized by **Robotis**.

The servomotor model is **Dynamixel AX-12.**

Illustration 42: Servomotor - Dynamixel AX-12

	AX-12	
Weight (g)	55	
Gear Reduction Ratio	1/254	
Input Voltage (V)	at 7V	at 10V
Final Max Holding Torque(kgf.cm)	12	16.5
Sec/60degree	0.269	0.196

Table 1: Servomotor - Dynamixel AX-12

Dynamixel servos are very versatile, quite strong and can be used in two modes:

- **Continuous Rotation** mode
- **Position** mode

The only limit of these servos causing some problem in this work is a **60° dead-band** .

When motor shaft falls in this range, position is not retrievable from the internal potentiometer . This was a problem in continuous rotation mode, because I could not control the servo position in that range. This limit **avoided** me to **synchronize** all the six whegs to obtain a perfect control in climbing over obstacles. In fact I ended up with a simple **open-loop motor control**.

The other 3 servos were used in **Position Mode**: Two servos for the **joint** in the center of robot One servo for the **tail**

In this case the dead band did not cause any problem, because a 180° movement was sufficient for the purpose.

I had to use **two** servos for the body joint, because of the **high torque** needed to move the head and the frontal pair of whegs.

At a first time I tried to use one servo, but it was subject to overheating, until it burned.

Using two servos in parallel, mechanically coupled, permitted a perfect control of the joint, without overheating.

4.2 Control Board

The central processing unit used is a **CM-510**.

It's a development board producted by Robotis, and its fully compatible with Dynamixel AX-12 servos.

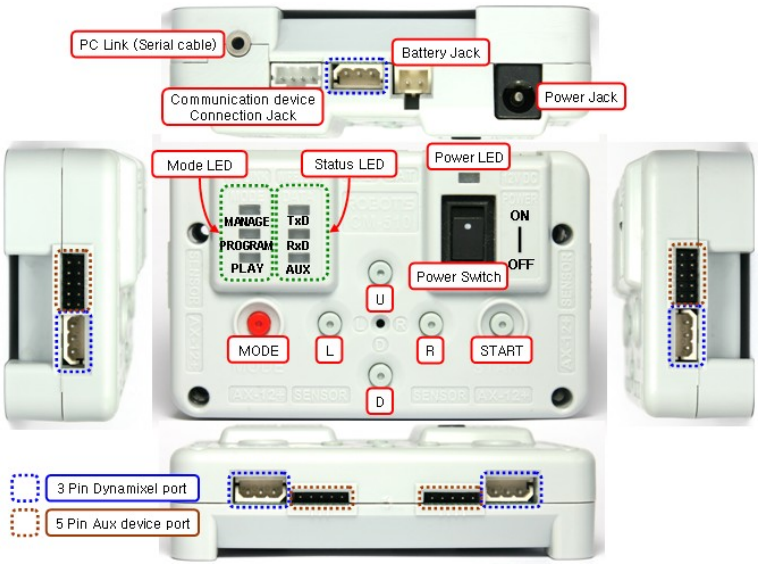


Illustration 43: Control Board - CM-510

	CM - 510
Weight	**51.3g**
Controller	**ATMega2561**
Working Voltage	• **Allowed Range : 6.5V ~ 15V** • **Recommended Voltage : 11.1V (Li-PO 3cell)**
Consumed Current	• **When IDLE : 50mA** • **External I/O Maximum Current : 0.9A** • **Total Maximum Current : 10A (Fuse)**

Table 2: Control Board – CM-510

This board is featured with standard **Dynamixel ports**, plus 6 other **general purpose ports**.

Each **general purpose port** is featured with 4 usable pins:

- Ground
- Power (5V)
- Digital I/O
- Analogue Input (0V– 5V)

I used these ports to connect all **peripheral sensors**.

Since I had too many sensors for the available ports, I had to built two **multiplexer** boards:

- One board to multiplex 4 rangefinders
- One board to multiple 3-axis accelerometer

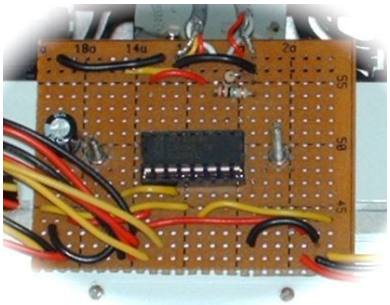


Illustration 45: Rangefinders Multiplexer Board

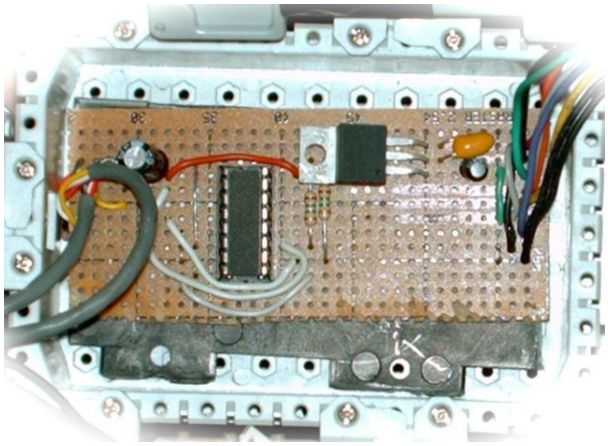


Illustration 44: Accelerometer Multiplexer Board

The boards inputs have not been fully used, so there are available channels for an eventual future use.

4.3 Sensors

The most critical aspects of perception in this robot are: Terrain sensing.

- Robot inclination with respect to gravity force vector.

4.3.1 Infrared Rangefinders

Terrain need to be analyzed in order to know if it's possible to walk over it or less.

If the terrain roughness is high, it is avoided: the robot turns left or right searching for a better path.

The key concept adopted to detect if a terrain is too rough is this: robot needs a planar surface in order to climb over it without possibility of falling.

In order to understand if terrain is planar, some IR rangefinders have been used.

This sensors are produced by SHARP, and the model is GP2D120X.

Illustration 46: Rangefinder - Sharp GP2D120X

Main specifications	GP2D120X
Measuring distance range	3 - 40 cm
Output terminal voltage	0.25 – 0.55
Average supply current	33 - 50 mA
Operating supply voltage	4.5 – 5.5 V

Table 3: Rangefinder - Sharp GP2D120X

The output distance characteristics of sensor is shown in the picture below.

As we can see , the characteristic is not linear.

In order to take correct distances (in cm) from the sampled values, I wrote a lookup table with 17 values and the corresponding tension value (represented as number)

```
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}
}
```

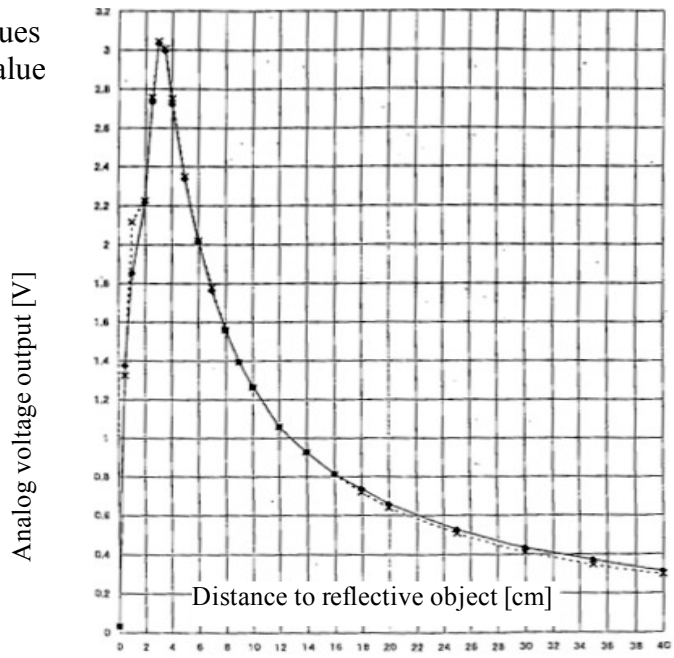


Illustration 47: Rangefinder Characteristic

The C function (used in the firmware code):

float readSharp_cm(unsigned char addr) interpolates the values in the lookup table, converting the tension value to distance in centimeters.

Four rangefinders are disposed onto a bar horizontally fixed on the head of the robot.

Three of these sensors point straight towards the terrain in front of robot, analyzing terrain's surface by sensing the distance from bar in three different points.

One sensor points in the march direction and is used to detect obstacles.

The sensors are disposed in this way:

- two in the center of the bar (one pointing down, one pointing forward)
- on in the left limit of the bar (pointing down)
- one in the right limit of the bar (pointing down)

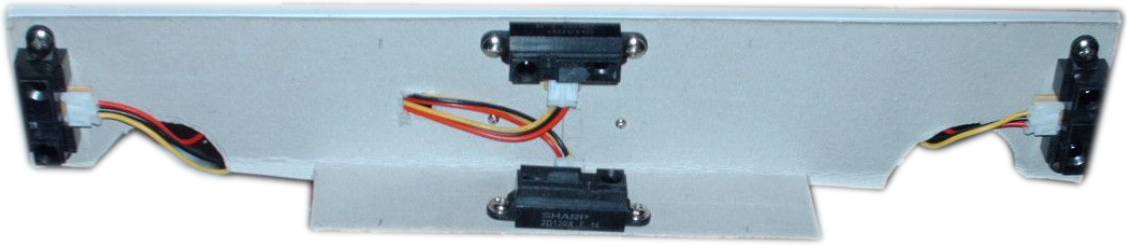


Illustration 48: Sensor Bar

Robot takes samples from all three sensors, and then compares them: If the distances are all the same (with a little tolerance), terrain is walkable. If the difference between right and center distances is high, robot turns slightly right If the difference between left and center distances is high, robot turns slightly left

Before using this approach, I tried to take distances with a single sensor turret, actuated by another Dynamixel servo, but this solution has revealed to be too slow and power consuming, and was abandoned.

As said before in the *Introduction* of this thesis, this sensor-bar has been conceived thinking at the **embodiment** concept: it moves with the robot body, giving automatically the sensors the best point of view (in order to have a good perception of terrain and obstacles) automatically. Sensor-bar position and orientation change according to the body configuration, which in turn changes according to the interaction with the environment.

4.3.2 3-axis Accelerometer

In order to know the robot inclination with respect to gravity force vector, a 3-axis accelerometer has been used.

This sensor is produced by **Freescale Semiconductor** and the model is **MMA7361** .

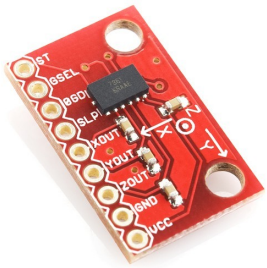


Illustration 49: Accelerometer - MMA7361

	MMA7361
Low Voltage Operation	2.2 V – 3.6 V
High Sensitivity	800 mV/g @ 1.5g
Selectable Sensitivity	(±1.5g, ±6g)
Low Current Comsumption	400 µA

Table 4: Accelerometer - MMA7361

I bought a ready to use *"breakout-board"* with sensor yet mounted on: you can see it in the photo above.

In this application the **selected sensitivity** is **±1.5g**

It has been mounted on the chassis, the most possible near to the center of mass of robot.

The robot center changes as the central joint moves, so a perfect positioning of the accelerometer has not been possible, but this was not a problem.

The accelerometer orientation is:

- x axis points towards robot forward sense of march, horizontally with respect to floor
- y axis is disposed horizontally with respect to floor, pointing to the left of robot
- z axis exactly in vertical position, pointing against gravity force vector.

The convention I used is to consider each axis in standard orientation when it is parallel

to the floor.

For each axis, the implemented controller computes the displacement_angle from standard orientation.

The method used to determine this displacement_angle is very simple: Each of three channels values are normalized with respect to the gravity force When robot is in standard position, values are 0 for y,x channels, 1 (100% gravity force) for the z channel.

1. The orientation of each axis is computed as:

displacement_angle = arcsin(channel_value_N)

where channel_value_N is channel value normalized to gravity module.

I'll explain this: each channel value is equal to cosin(gamma), where gamma is the angle between the axis and the gravity vector.

channel_value = g * cosin(gamma)

but we want to know the normalized value , so **channel_value_N = channel_value / g = cosin(gamma)**

The standard position is 90° wrt gravity vector, so the displacement_angle, we call it delta, is:

delta = 90° - gamma

so

gamma = 90° - delta

so

channel_value_N = cosin(90° - delta)

Knowing that

sin(delta) = cosin(90° - delta)

we can say

channel_value_N = sin(delta)

and finally

displacement_angle = delta = arcsin(channel_value_N)

4.4 Power

Robot has been powered by AC-DC transformer during the development phase.

Illustration 50: AC-DC Transformer

The transformer I used has this specifications: **Output Voltage 12V**

- **Max current 5A max**

When the robot development has been completed, robot has been equipped with a 2 cells **lithium polymer battery**.

Illustration 51: Li-PO Battery

Battery specifications:

- **Output Voltage 7.5V**
- **Battery capacity 2500 mAh**

Due to the lower tension of battery with respect to transformer, servo's speed has been slightly increased in order to ensure a sufficient torque.

Chapter 5 Firmware

The implemented firmware was initially conceived to be the only program to control the robot.

After some testing, the used microcontroller has revealed to be too less powerful in order to obtain an acceptable loop speed and a sufficiently complex computational capability. A complex control like motion planning needs a more performant computational system.

I decided to mantain a **fast reactive behavior based control** in the main loop, adding a serial communication phase in order to allow **data exchange** with external control hardware, like a PC. A deep analysis of the behavior based control in mobile robotics can be found in a recent work from L. De Silva and H. Ekanayake [18]: they discuss several behavior control paradigms, including the subsumption.

In remote control applications, a simple control like this will act as a cerebellum, assisting the remote user in the motion control. A recent work on this topic has been made by Shigang cui, Zhengguang lian, Li zhao, Zhigang bing, Hongda chen [19].

They discuss several behavior control paradigms, including the subsumption.

The implemented loop resulted to be very fast, and sufficient to manage the most critical behaviors, like falling prevention and obstacle climbing.

The main control loop consists of this macro blocks:

- **Read and Communicate Sensors Data**
- **Set Speed According to Body Inclination**
- **Main Behaviors**
  - **Jump Down Behavior**
  - **Terrain Check Behavior**
  - **Approaching Obstacle Behavior**
  - **Adapt to Floor Behavior**
- **Walking Actions**
  - **Go Backward**
  - **Turn Left**
  - **Turn Right**
  - **Restart Walking**

    - **Stop Walking**
- **Tail Control Section**
    - **Tail Behaviors Manager**
    - **Tail Behavior 1: Avoiding Falling Backward**
    - **Tail Behavior 2: Climbing**

5.1 Read and Communicate Sensors Data

The first thing to do in the main loop is reading sensors, and communicating them to the eventual external hardware through the serial-usb link.

5.1.1 Reading Sensors

This is done using this funcion:

- **void readAllSensors(Sensors *s);**

It reads all sensors data saving them into a ***Sensors*** structure.

The above function calls a specific read function for each sensor:

- **float readSharp_cm(unsigned char addr);**

It reads the value of **Sharp** sensor mapped at address ***addr*** and convert it to distance in **centimeters**.

Each address indicates a channel on the **distance sensors multiplexer.**

- **float readAccDeg(unsigned char addr);**

It reads the **angle** value from the **Accelerometer Channel** at address **addr**, and converts it to **degrees**.

Each address indicates a channel on the **accelerometer multiplexer**, and each multiplexer channel correspond to a single **channel** on the accelerometer.

Accelerometer, as seen before, has 3 channels, one for each **axis**.

5.1.2 Communicating sensor data

The communication is performed through a **serial link** in **asynchronous** mode, at a baudarate of **57600 bps**.

The serial format is **8bit, no parity, 1 stop bit**.

The data **format** is the following:

```
FC <FC>\tGC <GC>\tGL <GL>\tGR <GR>\tZ <Z>\tX <X>\tY <Y>\tAD <AD>\tTD
<TD>\n
```

and this is the meaning of the tags:

<FC> : *Distance in cm from Central Front Sensor*

<GC> : *Distance [cm] from Central Ground Sensor*

<GL> : *Distance [cm] from Left Ground Sensor*

<GR> : *Distance [cm] from Rigth Ground Sensor*

<Z> : *Normalized [adim] value from Z axis Channel on Accelerometer*

<X> : *Normalized [adim] value from X axis Channel on Accelerometer*

<Y> : *Normalized [adim] value from Y axis Channel on Accelerometer*

<AD> : *Abdomen Inclination angle [deg] with respect to the robot body*

<TD> : *Tail Inclination angle [deg] with respect to the robot body*

This data is intended to be read from the serial-usb interface using a **sscanf** function implemented in any programming language.

All transmitted values are integers (`%d` in C, C++ format).

This is the output we get connecting to it :

```
...
FC 39   GC 17   GL 19   GR 19   Z 932   X -24   Y 43    AD -34  TD -37
FC 39   GC 16   GL 18   GR 18   Z 1000  X 48    Y 92    AD -34  TD -37
FC 39   GC 15   GL 18   GR 19   Z 1000  X 54    Y 37    AD -34  TD -37
...
```

5.2 Set Speed According to Body Inclination

In order to adapt the servomotors torque to the current ground surface, the robot must increase their speed according to the vertical inclination of the robot body.

If the **inclination** value is high and **positive**, the robot is **walking uphill**, so the **torque** must be **increased**.

The following instruction sets the nominal speed for all whegs servos: ***dnspeed (Desidered Nominal Speed)*** :

```
dnspeed = (600 + 4 *sensors.body_inclination_xg);
```

Dynamixel servos accept a speed value from 0 to 1023.

Here we set 600 as base-speed, then adding a **variation** proportional to inclination.

If the inclination is **negative** (robot walking downhill) the added variation will be negative, ending up with a **speed value smaller than base-speed**.

5.3 Main Behaviors

This behaviors **determine** the commands that will be sent to the whegs in the following **Walking Actions** section.

All this behaviors are exclusive, and they have descending priority, so that the first has the high one and the last has the lower one.

Each behavior has a trigger event that I call "***situation***".

Before going to process the behaviors, some data must be prepared.

```
float ld = abs((int)(sensors.distance_floor_left –
sensors.distance_floor_center));
float rd = abs((int)(sensors.distance_floor_right –
sensors.distance_floor_center));
```

The above two instructions compute two difference values: **`ld`** : Left Difference

This is the difference value between the **left** terrain distance sensor and the

center terrain distance sensor

- **rd** : Right Difference
  This is the difference value between the **right** terrain distance sensor and the **center** terrain distance sensor.

This difference values will be used within the following behaviors in order to detect the current situation.

5.3.1 Jump Down Behavior

Situation 1: Robot's head is **facing floor** with an **abdomen inclination** inferior to **-45 °** with respect to **gravity force vector** AND **distance from floor** along the view axis is **less** than **30 cm**.

In this situation the robot will probably crash his head to the floor if it doesn't lift it instantly, so the first command is to stop walking.

This situation requires that the distance from floor is less than 30 cm, so that the robot could "jump" toward floor without destroying itself.

Behavior 1 : Then the robot lift up the abdomen until it exits from this situation.

```
if (sensors.abdomen_inclination_g < -45 && sensors.distance_front_center < 30){
   stop(); //STOP ROBOT
   while( sensors.abdomen_inclination_g < -45 &&
          sensors.distance_front_center < 30)
   {
          //LIFT ABDOMEN
          dpAbd = dpAbd + 1;
          setAbdomenDeg(dpAbd, 1000);//Lift
          sensors.abdomen_inclination_g = getAbdomenDegG();
          sensors.distance_front_center = readSharp_cm(4);
   }
}
```

5.3.2 Terrain Check Behavior

This behaviors uses the Left Difference (***ld***) and Right Difference (***rd***) described before in chapter 5.3.

Situation 2: Left Difference or Right Difference is greater than 8 cm.

This means that terrain is not uniform.

Behavior 2 : turn where the difference is smaller.

```
if(ld > 8 || rd > 8)
{
   if (ld >= rd) {turnR = 1; turnL=0;}
   if (ld < rd) {turnL = 1; turnR=0;}
}
```

5.3.3 Approaching Obstacle Behavior

Situation 3 : There is an obstacle near in front of the robot in central position.

Obstacle is considered **near** if distance is less than 15 cm.

Behavior 3 : **climb over** the osbtacle.

The sequence of actions to perform is :

1. stop walking
2. lift head (abdomen) until obstacle diappears and abdomen inclination doesn't exceed 80°. This limit has been imposed in order to avoid vain climbing efforts on almost perpendicular walls.

```
if (sensors.distance_front_center <  15)
{
   stop();
   while( sensors.distance_front_center < 15 && sensors.abdomen_inclination < 80)
       {
          if (dpAbd > 80)dpAbd = 80; //Upper Bound setAbdomenDeg(dpAbd, 200);//Lift up
          abdomen sensors.distance_front_center = readSharp_cm(4); //read distance
       }
}
```

5.3.4 Adapt to Floor Behavior

Situation 4 : Distance from floor is greater than 23 cm

This happens when the abdomen is too high, or when robot approaches a descent .

In order to have a good sensing of the terrain the sensors array should point the floor perpendiculary (and so abdomen should be parallel to terrain).

Due to the geometry of robot (his height is about 23 cm), lowering abdomen until it's 23 cm distant from floor will put it parallel to floor surface.

Behavior 4 : Lower the abdomen until distance from floor is less than 23 cm.

```
if(sensors.distance_floor_center > 23 ){ // Floor too distant ..

   stop(); //stop walking

   while( sensors.distance_floor_center > 23 && sensors.abdomen_inclination > -75)
       {
          dpAbd = dpAbd – 1;     // Lower down Abdomen
          if (dpAbd < -75)dpAbd = -75; // Limit value
          setAbdomenDeg(dpAbd, 200);//send command to servomotor

          sensors.distance_floor_center = readSharp_cm(6);
       }
}
```

5.4 Walking Actions

The purpose of this section is to send commands to servomotors, in order to perform the movements requested by the previous behaviors section.

The first instruction of this section is a control statement that checks if the robot should be walking or not.

This is accomplished by reading the value of a condition variable called ***walking***.

This condition variable is set within the previous behaviors section.

```
if (walking==1)
{
      …
}
```

The other condition variables used to control this section are:

```
TurnL //Turn Left
turnR //Turn Right
```

5.4.1 Go Backward

The way behaviors inform this section that the robot must bo backward is just enabling both turn condition variables : `turnL and turnR`.

```
If(turnL && turnR)
{
   int i ;
   for (i=0;i<3;i++){//Go Back
          dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L, 1424 );
          dxl_write_word( whegs_dx[i],  P_MOVING_SPEED_L, 400 );
          _delay_ms(20000);//Mantain behavior

          //Disable behavior..
          turnL = 0;
          turnR = 0;
   }
}
```

5.4.2 Turn Left

Turning Left is accomplished by going backward with the left train of whegs, keeping still the right one.

Behavior is mantained for a while, and then condition variable are cleared.

```
if(turnL)
{
   int i ;
   for (i=0;i<3;i++) //iterate on all servos    {
       dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L, 1424 );
       dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L, 0 );
```

```
          _delay_ms(20000);//Mantain behavior

          //Disable behavior..
          turnL = 0;
          turnR = 0;
   }
}
```

5.4.3 Turn Right

Same as Turn Left.

```
if(turnR){
   int i ;
   for (i=0;i<3;i++){
          //dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L, 400 );
          dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L, 0 );
          dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L, 400 );

          _delay_ms(20000);//Mantain behavior

          //Disable behavior..
          turnL = 0;
          turnR = 0;
       }
}
```

5.4.4 Restart Walking

This section is needed to make the robot restart walking in two cases:

- it was stopped before
- it was turning

```
   if (!(turnL + turnR)){//If not turning go_fwd();//Go forward
       }
```

5.4.5 Stop Walking

If the ***walking*** condition variable is not set, the robot just stops walking.

```
   If (walking==1)
{
```

```
      …
}
   else
      {
   stop(); //Stop Walking
       }
```

5.5 Tail Control Section

In this section we control the tail, in order to help robot to accomplish some tasks.

This part is located after the Walking Actions section, because it do not modifies any walking control variable.

5.5.1 Tail Behaviors Manager

This section acts like a behavior manager, it mantains a behavior until it accomplishes its task.

It checks if tails has reached (with little toelrance) the desidered position and, in positive case, it disables the torque on tail servomotor.

Then it disables all tail behaviors by clearing ***dpTail*** control variable.

```
// Check if desidered Tail position (if it was set) has been reached
if(dpTail != -1 && abs((int)dpTail - (int)pTail) < 50)
{
   dxl_write_word( 1, P_TORQUE_ENABLE, 0 );
    //Disable Torsion Servo
   // Needed to shut off tail behaviors //activated in previous loop cycle
   dpTail = -1; //disable desidered Tail position
}
```

5.5.2 Tail Behavior 1: Avoiding Falling Backward

Situation 5 :

- Tail is high
- Abdomen isn't high
- Body is tilted up more than 45° with respect to gravity vector

In this situation the robot is likely to fall backward, so robot body must be put in a less dangerous configuration.

Behavior 5 : Lower the tail until it's about in line with body (horizontal position)

Putting tail in line with body makes the back of the robot to lift up.

In this way robot's center of mass is moved forward, towards the central position, putting the whole body in a safer position.

```
if (   parallel && pTail < 300 &&  // Tail is high
       getAbdomenDeg() >-10 && // Abdomen isn't high
       readAccDeg(1) > 45) // Body is tilted up more than 45 deg wrt g
{
   //avoid falling backwards
   dpTail = 500;
   dxl_write_word( 1, P_GOAL_POSITION_L, dpTail ); dxl_write_word( 1, P_MOVING_SPEED_L,
   210 );
}
```

5.5.3 Tail Behavior 2: Climbing

Situation 6 :

- Tail is slightly up
- Abdomen is slightly low
- Body is tilted up more than 45° with respect to gravity vector

In this situation, robot's body is very tilted up, and abdomen is slightly tilted down.

This means that robot **is climbing over an obstacle**.

Behavior 6 : Lower the Tail until it's quite off from horizontal position.

If the tail is slightly upper than the horizontal position, it should be lowered down, in order to help robot in the climbing task.

Lowering tail helps because in this way the **robot's back is lifted up**, and this **moves** the **center of mass forward**, causing the frontal whegs to exert a greater **pressure** on terrain.

Great pressure on terrain ensures more **grip**, and so robot can easily bring its body forward, accomplishing the climbing task.

```
 if (  parallel && pTail < 450 &&     // Tail is slightly up
   getAbdomenDeg() < -20 &&      // Abdomen is slightly low
   readAccDeg(1) > 60 )  // Body is tilted up more than 60 deg wrt g
{      //climb
       dpTail = 712;
       dxl_write_word( 1, P_GOAL_POSITION_L, dpTail ); dxl_write_word( 1,
                       P_MOVING_SPEED_L, 210 );
}
```

5.6 Flow Chart

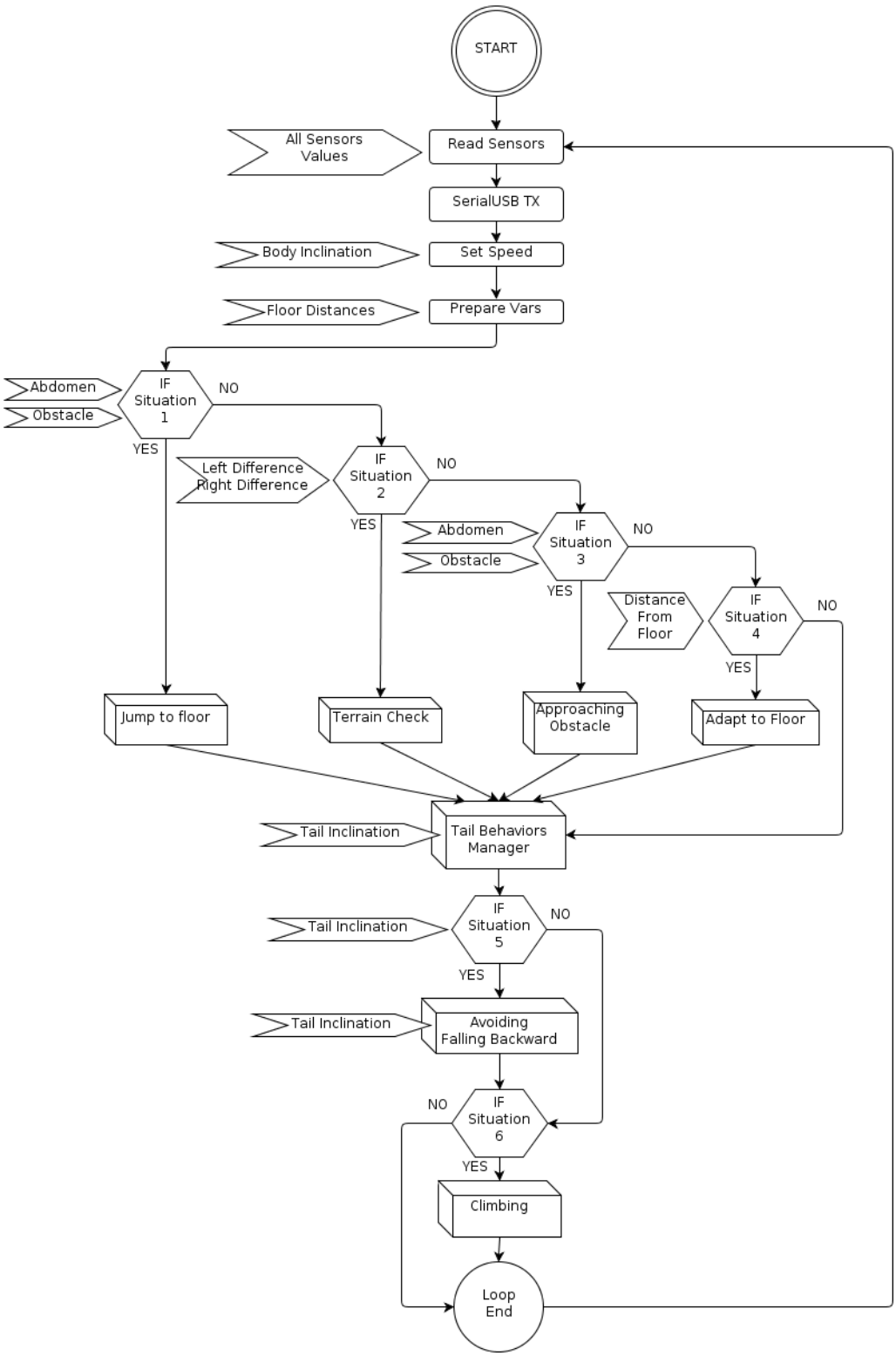


Illustration 52: Firmware - Main Control Loop Flow Chart

Chapter 6 Software

As said before, three plugins for the Player framework [15][16] have been implemented.

6.1 Player framework

Player is as server, and can interact with other software thru tcp/ip.

The host on which Player is installed is often located on the robot itself.

Other times , due to space and weight constraints, this host is not integrated into the robot, but runs on an external machine, connected to the robot in some way.

The Player server offers several **interfaces** to external programs that connects to it.

When an **external program** connects to the Player server, it **subscribes** to all interfaces it needs.

Each interface is bound to a **driver**.

Each driver is implemented into a **plugin**.

6.2 Serial Connection

Due to the small robot dimensions, I decided to run Player on an **external pc**, connected to the robot through serial-usb cable.

The **current** cabled **connection** system can be **replaced** in the future with a **zigbee radio** serial connection.

6.3 Three Plugins

The three implemented plugins are intended to be used on a linux environment, but only small modifications should be needed in order to adapt the code to other systems like

windows or osx.

These plugins have been implemented as **standard dynamic linux libraries**.

The programming language adopted is **C++**.

These plugins retrieve information from robot through **usb-serial link**, and then **publish** it into the **Player** framework, making it available to all eventual **subscribers**.

Players plugins can support several types of messages.

Current version of implemented plugins supports only this type of message:

PLAYER_MSGTYPE_DATA :

A data message.

Such messages are asynchronously published from devices, and are usually used to reflect some part of the device's state.

6.3.1 Position plugin

This plugin implements the ***position3d*** interface.

Position3d is a Player interface conceived to inform subscribers about the current position of robot in three dimension.

There is also a position2d interface, but I decided to use the 3d one because LionHell is intended to walk on very **rough** rurfaces, and the **tridimensionality** of **environment** is an **important** aspect.

The plugin is implemented into a library called: **libplayerLionHell_pos.so**

Current version is only a **partial** implementation.

In order to determine the robot position some **addtional sensors** are needed, like **gyroscope** or **GPS**.

The currently used **accelerometer alone** is **not sufficient** to **localize** the robot.

The **output** produced by this plugin, obtained connecting to it with the client tool

playerprint (provided by the Player project) , is this:

```
#Position3d (30:0)
#X      Y       Z       Roll    Pitch   Yaw
Pos: 0 0 0 0 0 0
Vel: 0 0 0 0 0 0
```

6.3.2 Ranger Plugin

This plugin implements the ***ranger*** interface.

Ranger is a Player interface conceived to inform subscribers about the **current distance values** of one or more **rangefinders** mounted on a robot.

The plugin is implemented into a library called: **libplayerLionHell_ranger.so**

This interface provides even other information relative to each mounted rangefinder sensor:

- **geometry** of sensor.
- **position** of sensor.
- **orientation** of sensor.
- **configuration** of sensor

The current version of plugin supports the following information: **ranger values**

- **geometry**
- **position**
- **orientation**

Plugin **output** is this:

```
#Ranger (62:0)
Device pose: (0.296845, 0, -0.0188897), (0, -42, 0) Device size: (0.015, 0.045, 0.013)
Configuration:
Minimum angle: 0       Maximum angle: 0       Angular resolution: 0 Minimum range: 0
        Maximum range: 0Range resolution: 0 Scanning frequency: 0
4 range readings:
[32, 14, 15, 17]
```

6.3.3 Robot Plugin

This plugin is a **wrapper** to the other two plugins, and its purpose is to **group** all plugins into one, permitting an **easy** way to acces all **robot functionalities** from the Player server.

The interfaces offered by this plugins are:

- *position3d*
- *ranger*

The plugin is implemented into a library called: **libplayerLionHell_robot.so**

The **output** is the **same** of the other two plugins.

As we can see in the *Software Code* section in the *Appendix C* of this thesis, this plugin connects to the other two, retrieving information continuously and publishing it to all the eventual subscribers.

Chapter 7 Experiments

In this section I'll show you the robot behavior indoor and outdoo**r.**

7.1 Walking upstairs indoor

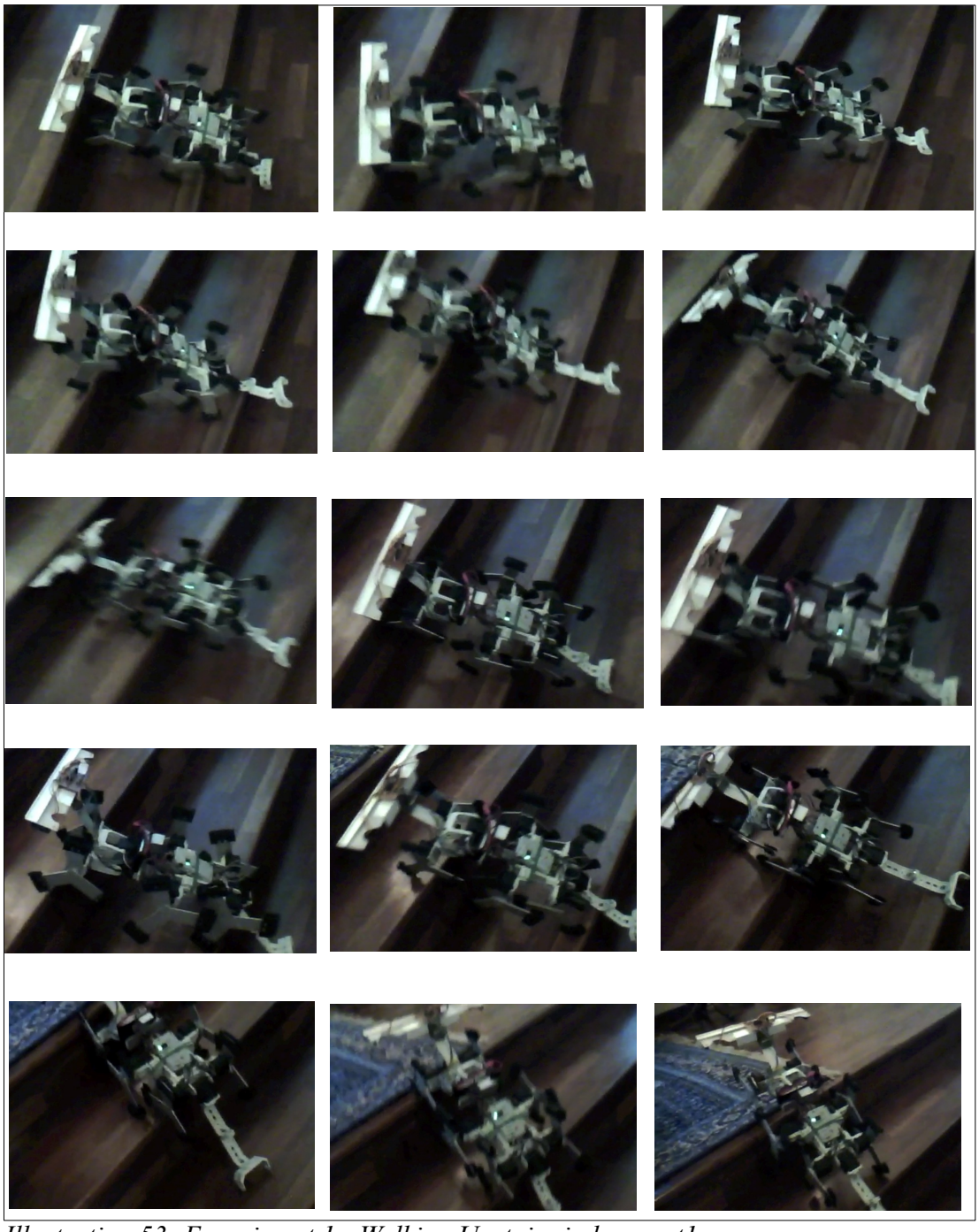


Illustration 53: Experiment 1 - Walking Upstairs indoor - pt1

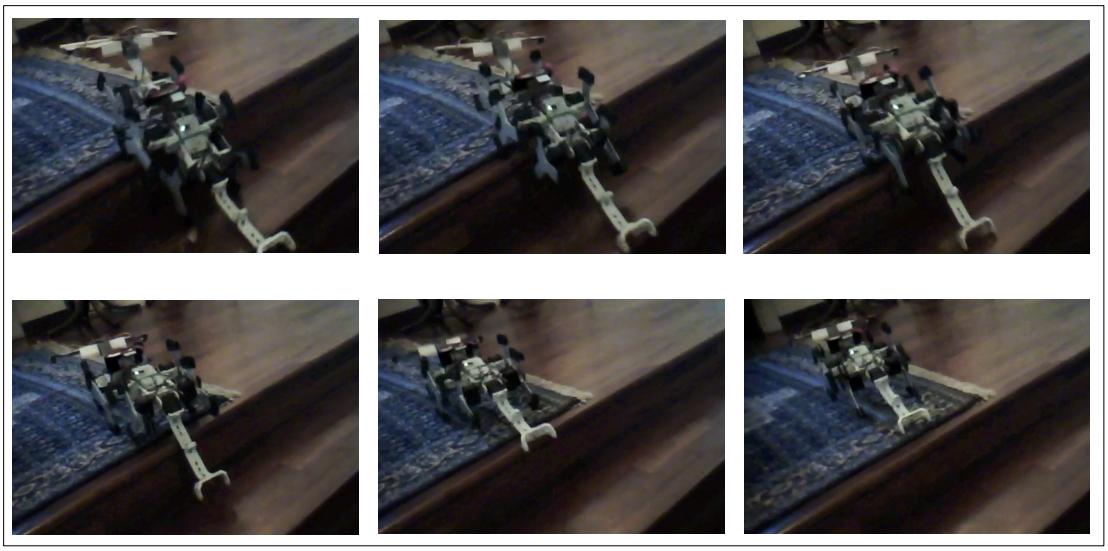


Illustration 54: Experiment 1 - Walking Upstairs indoor - pt2

In the stair-climbing task, five behaviors acts iteratively: *Approaching Obstacle*, *Adapt to Floor*, *Climb* and *Avoid Falling Backwards*

7.2 Overcoming a big obstacle indoor

Illustration 55: Experiment 2 - Overcoming a big obstacle indoor - pt1

Active behavior: *Approaching Obstacle*

Illustration 56: Experiment 2 - Overcoming a big obstacle indoor - pt2

Active behavior: *Adapt to Floor*

Illustration 57: Experiment 2 - Overcoming a big obstacle indoor - pt3

Active Behavior: *Climb*

Illustration 58: Experiment 2 - Overcoming a big obstacle indoor - pt4

Active behavior: *Jump Down*

Illustration 59: Experiment 2 - Overcoming a big obstacle indoor - pt5

7.3 Climbing over a big obstacle outdoor

Illustration 60: Experiment 3 - Climbing over a big obstacle outdoor - pt1

Here the active behavior is *Approaching Obstacle*

Illustration 61: Experiment 3 - Climbing over a big obstacle outdoor - pt2

Abdomen is lowered due to the *Adapt to Floor* behavior, in order to touch the obstacle surface with frontal whegs, and **tail** is lifted up by the *Climb* behavior.

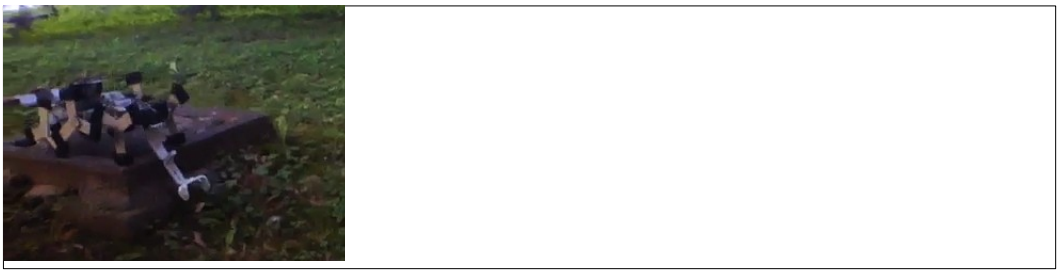


Illustration 62: Experiment 3 - Climbing over a big obstacle outdoor - pt3

7.4 Overcoming a small obstacle outdoor

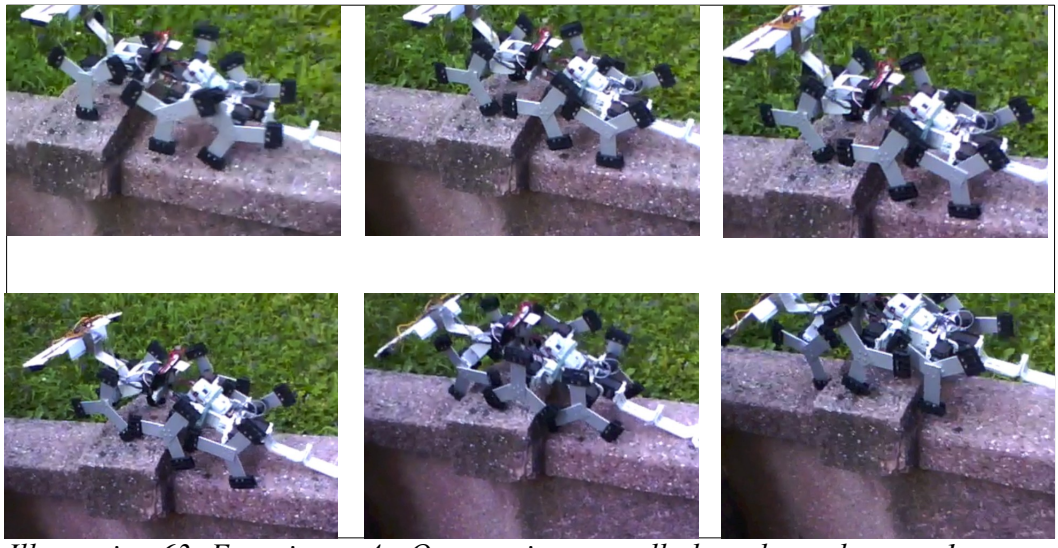


Illustration 63: Experiment 4 - Overcoming a small obstacle outdoor - pt1

Robot slightly lower down the abdomen due to the *Adapt to Floor* behavior.

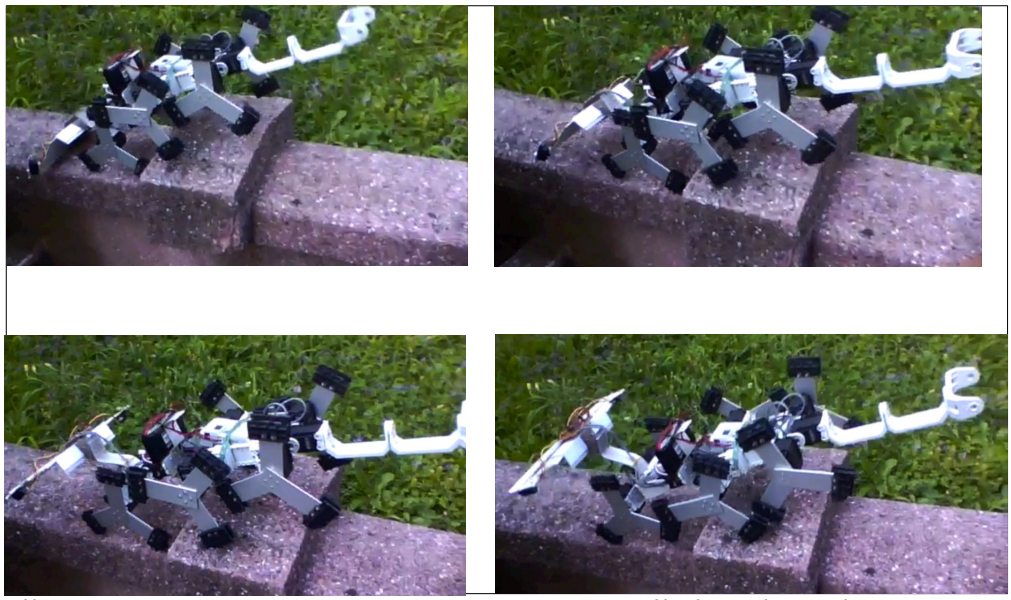


Illustration 64: Experiment 4 - Overcoming a small obstacle outdoor - pt2

In the last part of this experiment the robot lift slightly up its abdomen, due to the *Adapt to Floor* behavior.

7.5 Walking up stairs outdoor

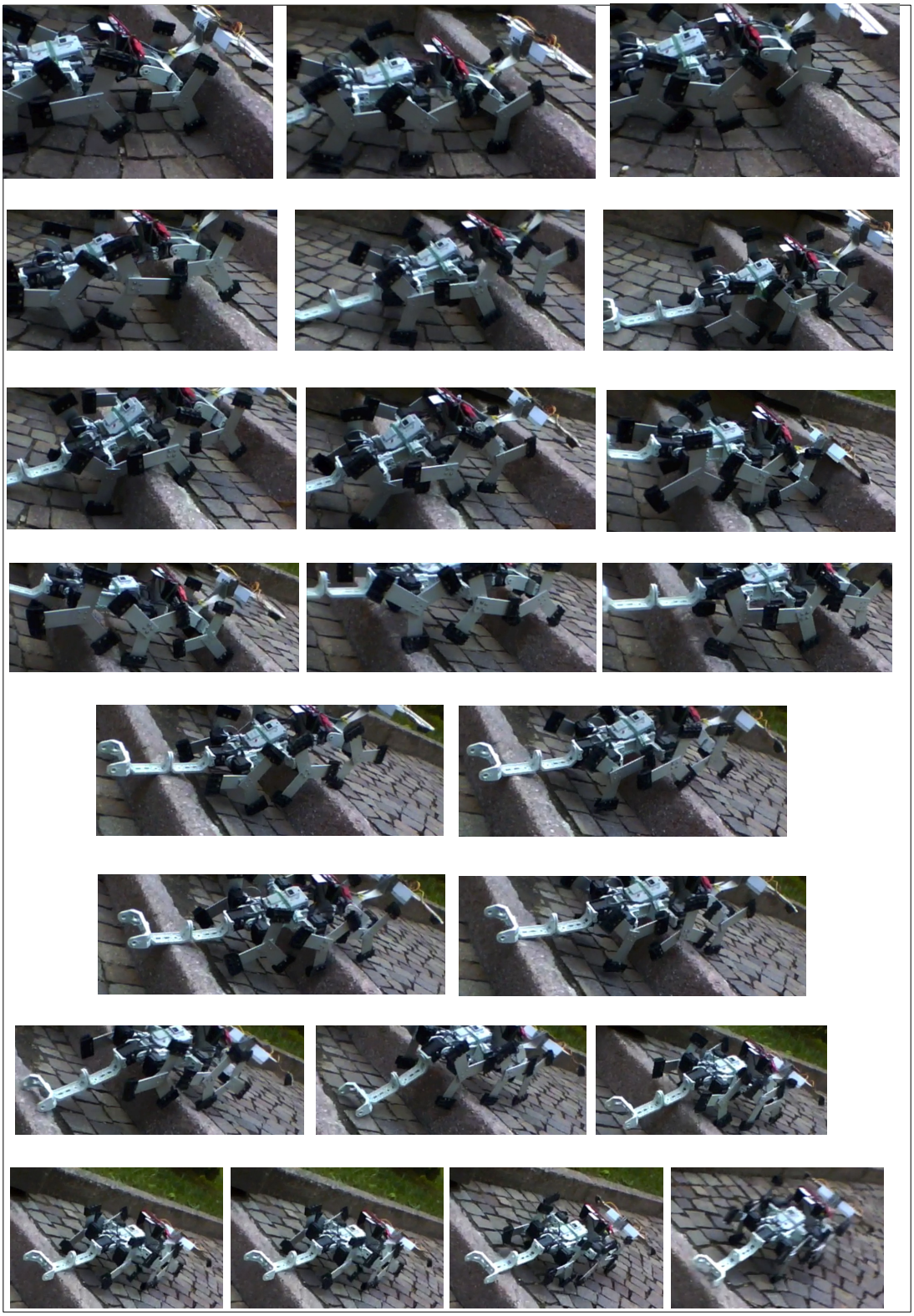


Illustration 65: Experiment 5 - Walking up stairs outdoor

Behaviors: *Approaching Obstacle*, *Adapt to Floor*, *Climb, Avoid Falling Backwards*

7.6 Walking down obstacle outdoor

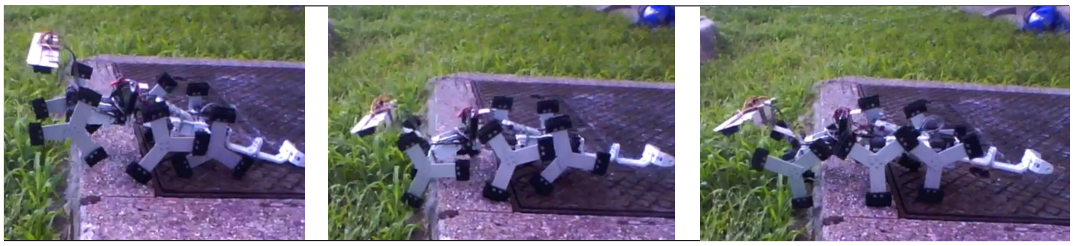


Illustration 66: Experiment 6 - Walking down obstacle outdoor - pt1

Robot flex its abdomen down during the first part of the descent: this is due to the *Adapt to Floor* behavior *(figure above)*.

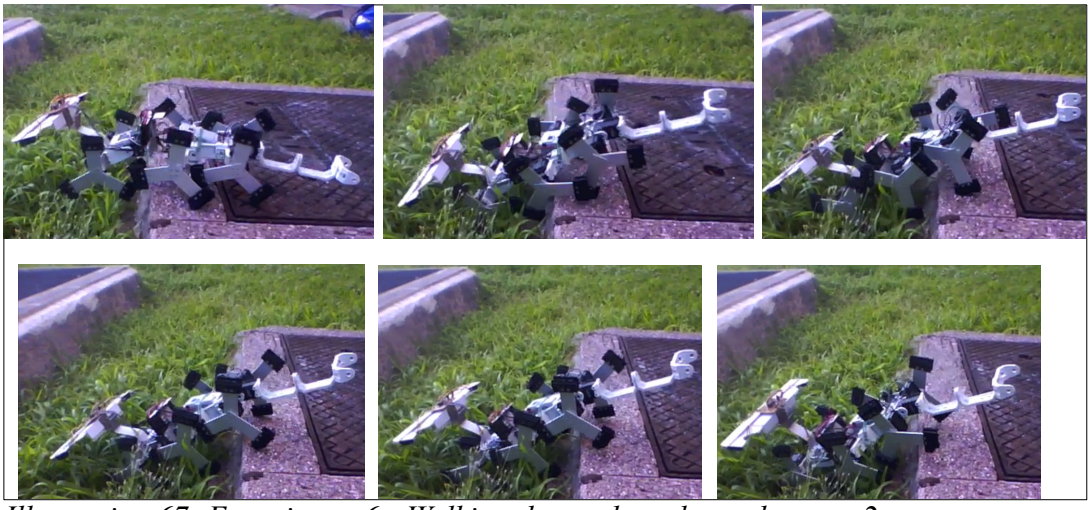


Illustration 67: Experiment 6 - Walking down obstacle outdoor - pt2

In the second part of the descent robot lifts up its abdomen: it's hard to understand the involved behavior. It could be both *Adapt to Floor* both *Approaching Obstacle*.(figure above)

Illustration 68: Experiment 6 - Walking down obstacle outdoor - pt3

7.7 Walking uphill outdoor

Illustration 69: Experiment 7 - Walking uphill outdoor

Robot reaches the top of the small hill . No behaviors have been activated.

7.8 Walking on rough terrain

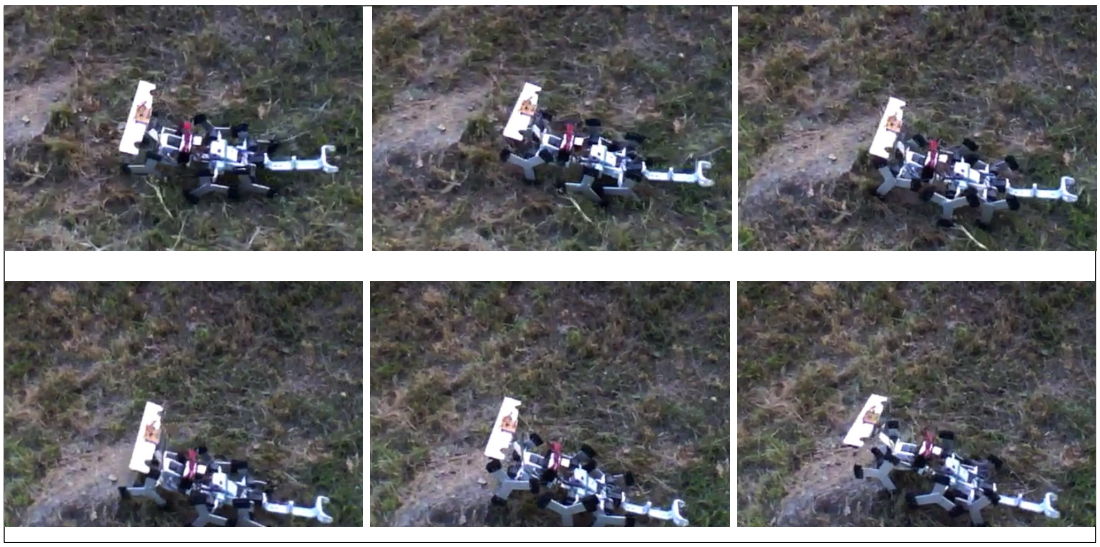


Illustration 70: Experiment 8 - Walking on rough terrain - pt1

Please note the surface unevenness: it acts as an obstacle. Robot is able to overcome it. Behaviors involved: first *Approaching Obstacle* and then *Adapt to floor*.

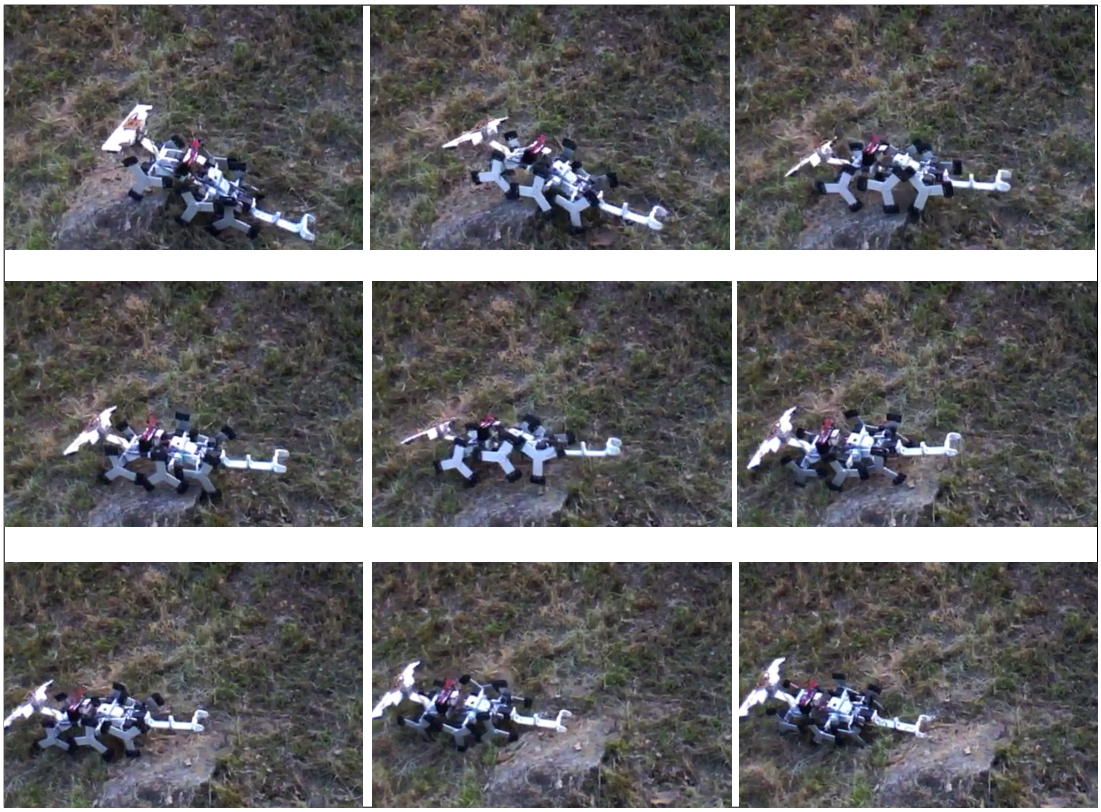


Illustration 71: Experiment 8 - Walking on rough terrain - pt2

7.9 Walking on natural terrain

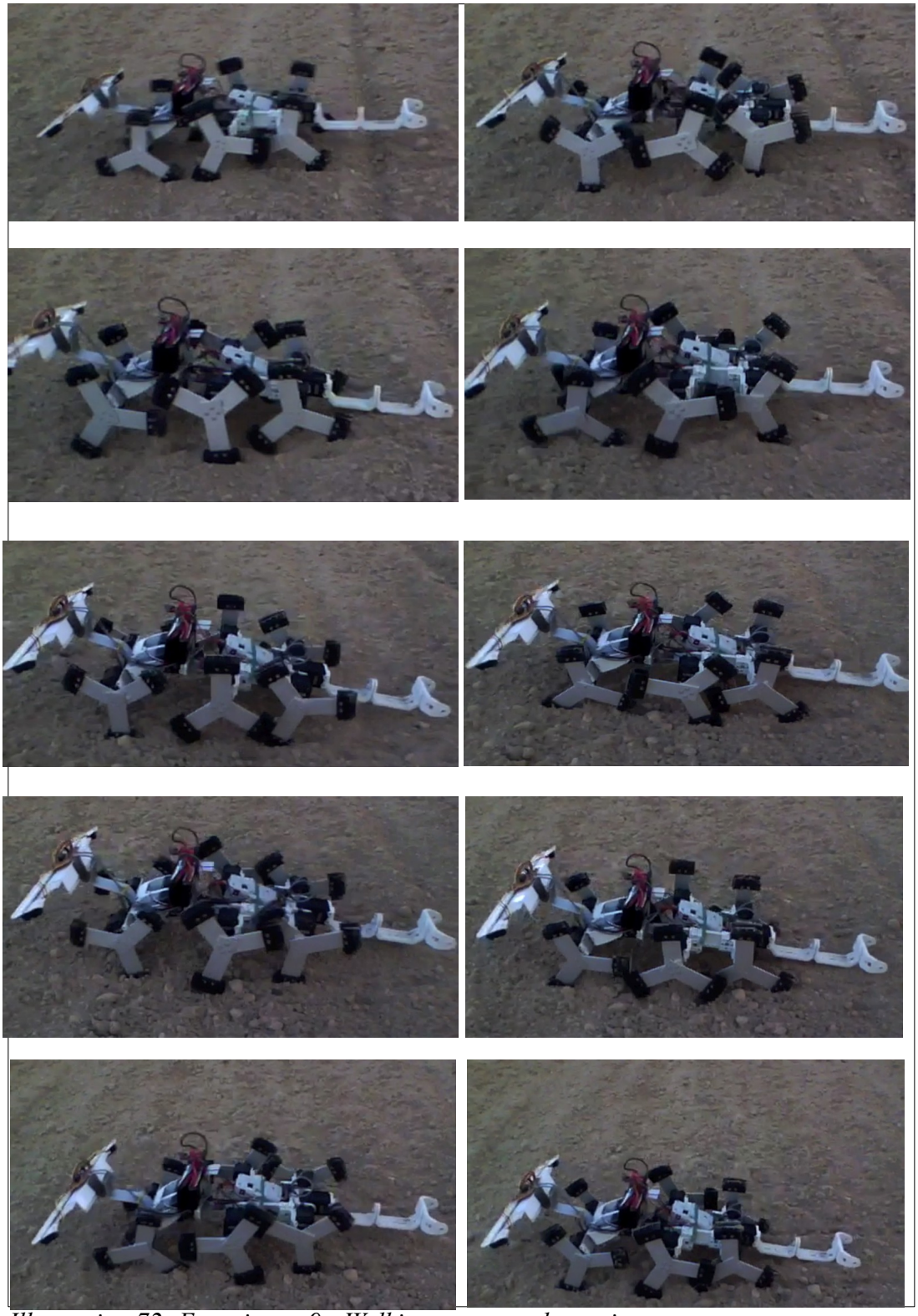


Illustration 72: Experiment 9 - Walking on natural terrain

Robot walks without any problem.

7.10 Walking on wild terrain

Illustration 73: Experiment 10 - Walking on wild terrain - pt1

Illustration 74: Experiment 10 - Walking on wild terrain - pt2

Robot walks without any problems for a while, then, in this frame, the robot is completely **stuck**. **Wheg system** has problems in these kind of surfaces: due to the **rotational movement**, **legs** easily get **entagled** in the weed.

Chapter 8 Conclusions and Future Work

The **designed** and **implemented** robot structure is able to **overcome several types** of **obstacles**, which can be found on **rough terrain environments**.

The **developed system**, based on **morphological computation** [1], permits robot to adapt to terrain using **tail** and **body joint.**

The new locomotion system , called **wheg**, has several **benefits** with respect to **wheels** or **legs**.
Whegs are poor on rough terrain, but very easy to control.
Legs are highly adaptive to terrain, but they need too much control complexity, and need more then one actuators.
Whegs combine both of these features into a single component that's easy to build and to control.

The new wheg-robot model developed has more degrees of freedom than the previous wheg robots [2][3][4][5][6][7][8][9][10][11], it's the wheg robot with more sensors on board and this is the first wheg robot using both tail both body-joint features.
In other wheg robots only one of these features at time has been adopted, but never both simultaneously.
On the Ratasjalg robot only the tail has been used , and it's not actuated [11].
In Whegs II , in Climbing-miniwhegs B31 and in Climbing-miniwhegs B00 [9] , only body joint has been used.

The task of **overcoming** big **obstacles** is accomplished with **minimal control complexity**.

The **current** robot control **does not** permit a perfect management in all **situations**, and this is due to the **limited** elaboration **complexity** of **sensor** data, other than the limited **number** of sensors used.

The specific limit is the **poor terrain perception** that makes hard for robot to **choose** a **safe path** to walk on.

Another aspect to be improved is robot stability: using **orientation** data (with respect to **gravity**) from **accelerometer,** a **short-range path planning** algorithm focused on robot **stability** can be developed.

Using the **Player** framework it is possible to develop a more **complex** control system capable of managing a **broader** set of **situations [15][16]**.

Future work can **extend** the implementation of Player's **plugin**, in order to develop a **control system** on a **higher level of abstraction**.

Another possibility is to develop **swarm** applications: the used servomotors could be easily **replaced** with **cheaper** models (digital or analog servomotors with continuous rotation feature) , ending up with a **low cost wheg robot** and permitting to build a **fleet** of robots at a **small price**.

Recent studies have been made on swarm applications in rough terrain environments [20][21].

Bibliography

1 Pfeifer, R. and Hoffmann M., *Embodiment, morphological computation, and robot body schemas*,(Zurich, 2010)

2 Quinn, R. D. Nelson, G.M., Bachmann, R.J., Kingsley, D.A., Offi, J. And Ritzmann, R. E., *Insect Designs for Improved Robot Mobility.*,In Proc. Of Climbing and Walking Robots Conference (CLAWAR01)(Germany, 2001)

3 Lewinger, W.A., C.M. Harley, R.E. Ritzmann, M.S. Branicky, and R.D. Quinn, *Insect-like Antennal Sensing for Climbing and Tunneling Behavior in a Biologically-inspired Mobile Robot.*, IEEE International Conference on Robotics and Automation (ICRA'05) Proceedings(Barcelona, Spain, 2005)

4 Lewinger, W.A., M.S. Watson, and R.D. Quinn, *Obstacle Avoidance Behavior for a Biologically-Inspired Mobile Robot Using Binaural Ultrasonic Sensors*,(Beijing, China,)

5 Jeremy M. Morrey, Bram Lambrecht, Andrew D. Horchler, Roy E. Ritzmann, Roger D. Quinn, *Highly Mobile and Robust Small Quadruped Robots*,(Cleveland, Ohio, U.S.A.,)

6 Daltorio K.A., Horchler A.D., Gorb S., Ritzmann R.E. And Quinn R.D. , *A Small Wall-Walking Robot with Compliant, Adhesive Feet*,Int. Conf. On Intelligent Robots and Systems (IROS '05)(Edmonton, Canada, 2005)

7 Daltorio K.A., Gorb S., Peressadko A., Horchler A.D., Ritzmann R.E. And Quinn R.D., *A Robot that Climbs Walls using Micro-structured Polymer Feet*,International Conference on Climbing and Walking Robots (CLAWAR)(London, U.K., 2005)

8 Kathryn A. Daltorio, Terence E. Wei, Stanislav N. Gorb, Roy E. Ritzmann, Roger D. Quinn, *Passive Foot Design and Contact Area Analysis for Climbing Mini-Whegs*,Proceeding of the IEEE International Conference on Intelligent Robots and Systems (ICRA'07) (Roma, Italy, 2007)

9 Kathryn A. Daltorio, Timothy C. Witushynsky, Gregory D. Wile, Luther R. Palmer, Anas Ab Malek, Mohd Rasyid Ahmad, Lori Southard, Stanislav N. Gorb, Roy E. Ritzmann, Roger D. Quinn, *A Body Joint Improves Vertical to Horizontal Transitions of a Wall-Climbing Robot.*,Proceeding of the IEEE International Conference on Intelligent Robots and Systems (ICRA'08) (Pasadena, CA, U.S.A., 2008)

10 A. Martin-Alvarez, W. De Peuter, J.Hillebrand, P.Putz, A.Matthyssen, J.F. de Weerd, *Walking Robots for Planetary Exploration Missions*,WAC'96, Second World Automation Congress(Montpellier, France, 1996)

11 Raivo Sell, Mati Kaeeli, *Invention Patent :Wheel Stand,*,priority number: P200700027, priority date: 01.06.2007(, 2007)

12 Hitoshi KIMURA, Shigeo HIROSE, *Development of Genbu : Active wheel passive joint articulated mobile robot*,Proceedings of the 2002 IEEURSJ Intl. Conference on Intelligent Robots and Systems EPFL(Lausanne, Switzerland, 2002)

13 Zhiqing Li, Shugen Ma, Bin Li, Minghui Wang and Yuechao Wang, *Design and Basic Experiments of a Transformable Wheel-Track Robot with Self-adaptive Mobile Mechanism*,The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems(Taipei, Taiwan, 2010)

14 J. Suthakorn, S.S.H. Shah , S. Jantarajit , W. Onprasert and W. Saensupo, S. Saeung, S. Nakdhamabhorn,V. Sa-Ing, and S. Reaungamornrat, *On the Design and Development of A Rough Terrain Robot for Rescue Missions*,Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics (Bangkok, Thailand, 2009)

15 Toby H.J. Collett, Bruce A. MacDonald, and Brian P. Gerkey, *Player 2.0: Toward a Practical Robot Programming Framework*,Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005) (Sydney, Australia, 2005)

16 Brian Gerkey, Richard T. Vaughan and Andrew Howard, *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems*,In Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)(Coimbra, Portugal, 2003)

17 Hyungseok Kim, Taehun Kang, Vo Gia Loc and Hyouk Ryeol Choi, *Gait Planning of Quadruped Walking and Climbing Robot for Locomotion in 3D Environment*,Proceedings of the 2005 IEEE International Conference on Robotics and Automation(Barcelona, Spain, 2005)

18 L. De Silva and H. Ekanayake, *Behavior-based Robotics And The Reactive Paradigm, A Survey*,Proceedings of International Workshop on Data Mining and Artificial Intelligence (DMAI' 08) (Khulna, Bangladesh, 2008)

19 Shigang cui, Zhengguang lian, Li zhao, Zhigang bing, Hongda chen, *Design and Path Planning for a remote-brained Service Robot*,Third International Conference on Natural Computation (ICNC 2007)(Haikou, Hainan, China, 2007)

20 Schmickl Thomas, Möslinger Christoph, Thenius Ronald, Crailsheim Karl, *Bio-inspired Navigation of Autonomous Robots in Heterogenous Environments*,(,)

21 Alejandro Ramirez-Serrano, Giovanni C. Pettinaro, *Navigation of Unmanned Vehicles Using a Swarm of Intelligent Dynamic Landmarks*,Proceedings of the 2005 IEEE International Workshop on Safety, Security and Rescue Robotics (Kobe, Japan, 2005)

Web Resources

Builded Robot (*LionHell McMillan*)

Project page on AIRWiki :

http://airlab.elet.polimi.it/index.php/LionHell_McMillan

Wiki page on RobotGarage.org :

http://www.robotgarage.org/wiki/index.php?title=LionHell_McMillan

Datasheets

Servomotors
Model: Dynamixel AX-12
Manufacturer: Robotis
http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm

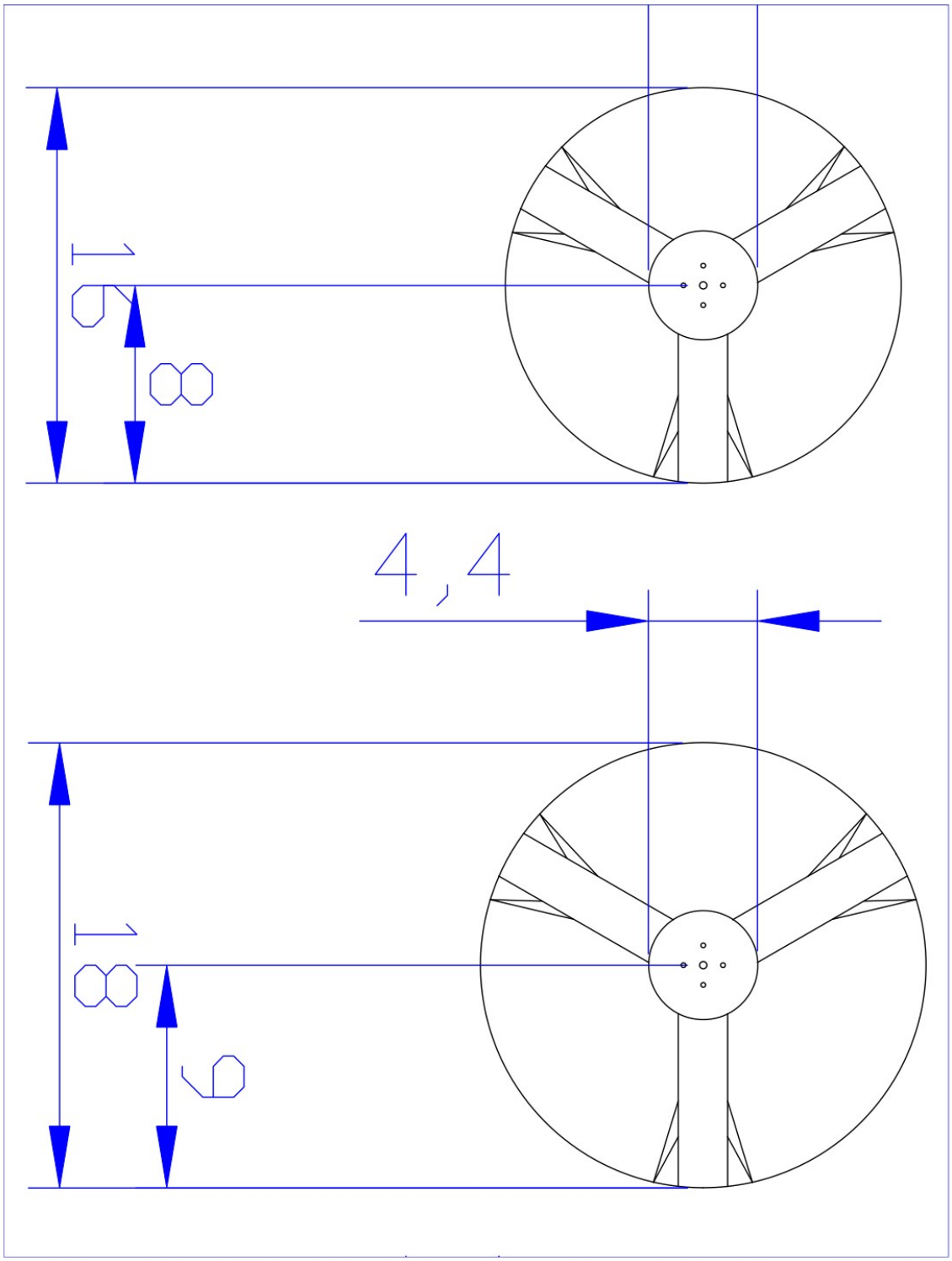
Control Board
Model: CM-510
Manufacturer: Robotis
http://support.robotis.com/en/product/auxdevice/controller/cm510_manual.htm

Microcontroller on Control Board
Model: ATMega2561
Manufacturer: Atmel
http://www.atmel.com/Images/2549S.pdf

IR rangefinders
Model: GP2D120X
Manufacturer: Sharp
http://www.sharpsma.com/webfm_send/1205

Accelerometer
Model: MMA7361L
Manufacturer: Freescale Semiconductor
http://www.freescale.com/files/sensors/doc/data_sheet/MMA7361L.pdf

Appendix A : Wheg Mechanical Drawing



Drawing 1: Wheg Mechanical Drawing

Appendix B : Firmware

An updated version of this firmware can be downloaded from the LionHell McMillan page hosted on RobotGarage.org (see the Web Resources Chapter).

This code is intended to be used on CM-510 board.
Required libraries and include files can be downloaded from Robotis website under the section *Embedded C* :
http://support.robotis.com/en/software/embeded_c/cm510_cm700.htm

The compiler used is AVRGCC.
The system on which has been developed is Mac OS X 10.5.

Embedded C Code
Text 1: Firmware Code

```
/*
 * AVRGCC1.c
 *
 * Created: 24/11/2011 20.33.13
 *  Author: venom
 */

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include "dynamixel.h"
#include "serial.h"
#include <math.h> //libm.a (doesn't work for me in winavr, but works in avr-gcc).

//Time Vars
#define TIME_SEC             15625
#define TIME_DECSEC          1563
#define TIME_CENTSEC                    156
#define TIME_MILLISEC                   16

// AX-12+ Control table addresses
#define P_GOAL_POSITION_L               30
#define P_GOAL_POSITION_H               31
#define P_PRESENT_POSITION_L            36
#define P_PRESENT_POSITION_H            37
#define P_PRESENT_SPEED_L               38
#define P_PRESENT_SPEED_H               39
#define P_PRESENT_LOAD_L                40
#define P_PRESENT_LOAD_H                41
#define P_MOVING                        46
#define P_MOVING_SPEED_L                32
#define P_MOVING_SPEED_H                33
#define P_TORQUE_ENABLE                 24

// AX S1 Control table addresses
#define LEFT_IR_SENSOR_DATA             26
#define CENT_IR_SENSOR_DATA             27
#define RIGH_IR_SENSOR_DATA             28
#define OBSTACLE_DETECTION_FLAG         32
#define OBSTACLE_DETECTED_COMPARE_VALUE 20 //Obstacle Detection Threshold
#define OBSTACLE_DETECTED_COMPARE       52 // 0: Low Dist Sens, 1: High Dist Sens
#define CENTER_LUMINOSITY               30
```

```
//Button defines
#define BTN_UP 0x10
#define BTN_DOWN 0x20
#define BTN_LEFT 0x40
#define BTN_RIGHT 0x80
#define BTN_START 0x01

// Default setting
#define DEFAULT_BAUDNUM                1 // 1Mbps
#define DEFAULT_ID                     1

struct ServoData{
       int id;
       unsigned short lastP[7];//Master Servo: Present Position data
       _Bool db;//Dead Band state: 0 = not in db; 1 = in db;
       unsigned short spd; //Master Servo: Present Speed Data
};

//
// Structure used to containthe parameters of the IR distance sensors
//
typedef const struct
{
       const signed short a;
       const signed short b;
       const signed short k;
}
ir_distance_sensor;


/* All sensor values */
typedef struct
       {
          float distance_front_center; //= readSharp_cm(4);
          float distance_floor_center; //= readSharp_cm(6);
          float distance_floor_left; //= readSharp_cm(5);
          float distance_floor_right; //= readSharp_cm(7);
          short body_inclination_xg; //= readAccDeg(1);
          short body_inclination_yg; //= readAccDeg(2);
          short body_inclination_zg; //= readAccDeg(0);
          short abdomen_inclination; //= getAbdomenDeg();
          short abdomen_inclination_g; //= getAbdomenDegG();
       }
       Sensors;

void init_OC1A(void);
void initTimer1(void);
unsigned int Timer1Read( void );
void Timer1Write( unsigned int i );
void Timer1Pause(unsigned int flag);/*Pause using timer 1 (warning: it resets timer!) */
void PrintCommStatus(int CommStatus);
void PrintErrorCode(void);
unsigned char readAXS1cd(void);
void go_fwd(void);
void stop(void);
void torque(unsigned char val);
int checkDxlConn(void);
void adc_init(void);
uint16_t adc_start(unsigned char channel);
float ir_distance_calculate_cm(ir_distance_sensor sensor, unsigned short adc_value);
unsigned short mirror(unsigned short speed);/* Mirror Speed: 0 to 1023, 1024 to 2047 */
unsigned short mirrorPos(unsigned short pos);/* Mirror Position: 0 to 1023 */
unsigned short pos2deg(unsigned short pos); /* deg: 0 to 300 */
unsigned short deg2pos(unsigned short deg); /* deg: 0 to 300 */
short  getTailDeg(void);
void setTailDeg(short deg, unsigned short speed);
short  getAbdomenDeg(void);/* deg: -75 to +80 */
short  getAbdomenDegG(void); /* Gets inclination wrt g-plane. deg: -150 to +150 */
void setAbdomenDeg(short deg, unsigned short speed); /* deg: -75 to +80 */
unsigned short dist2cm(unsigned short dist);
float cosML(float x);
float readHeight(void);
void frontLine_READ_ARRAY(void);
void setupAcc(void);/* Configure MCU bits to use Accelerometer */
uint16_t readAcc(unsigned char addr);
float readAccNorm(unsigned char addr);/* read calibrated and normalized (0-1) value from Accel Channel @ addr */
float readAccDeg(unsigned char addr);/* read deg value from Accel on channel @ addr */
void setupSharp(void);/* Configure MCU bits to use Sharp IR Sensors Board */
uint16_t readSharp(unsigned char addr); /* Read value of Sharp sensor mapped on address "addr" */
float readSharp_cm(unsigned char addr); /* Read value of Sharp sensor mapped on address "addr"  and convert to cm
```

```
distance */
void readAllSensors(Sensors *s); /* Read all sensor values and put into struct s */
void printAllSensors(Sensors sensors);

//// Globals ------------------------

//Communication Dynamixel
_Bool comm_err = 0; //Used to ensure correct communication with dynamixel
//SHARP Sensor
// This var contains the parameters of GP2Y0A21YK sensor
const ir_distance_sensor GP2Y0A21YK = { 5461, -17, 2 };

//Communication
int CommStatus;

//Whegs
int whegs_sx[3] = {6,3,14};
int whegs_dx[3] = {8,7,12};

//Desired Nominal Speed
unsigned short dnspeed = 0;//280;

struct ServoData sdata[7];
unsigned char dCIR; //Center IR Sensor Distance
unsigned char dLIR; //Left IR Sensor Distance
unsigned char dRIR; //Right IR Sensor Distance
unsigned char lCIR; //Center IR Sensor Luminosity
_Bool parallel = 1; //Parallel Mode for Whegs
int parallel_time = 0;//Parallel Config Time var
int frontLift_time = 0;//FrontLift Config Time var
int whegBlocked_time = 0;//Wheg Blocked Time var
_Bool wheg_blocked = 0 ; //Wheg Blocked state
short dpAbd = 0;//Desired Abdomen Inclination wrt Body in [deg]
unsigned short pTail ;//Tail Servo Position
int dpTail = -1; //-1 : no desidered position (any is good)
_Bool dir15 = 0; //sensor servo (id 15) direction. 0 = CW, 1= CCW.
Sensors sensors;    /* Holds all sensors values */
float x_speed = 0; /* X direction speed , 2 be calc'd from x_acc */

//Button vars
_Bool btn_switch1 = 0;
_Bool btn_switch2 = 0;


//Behaviors Controllers
unsigned short turnL = 0;
unsigned short turnR = 0;
_Bool walking = 1;



int main(void)
{
       serial_initialize(57600);
       dxl_initialize( 0, DEFAULT_BAUDNUM ); // Not using device index
       sei();       // Interrupt Enable
       printf( "\n\n Venom Whegs Sync Master\n\n" );
       _delay_ms(20000);
       // Enable Continuous Turn on whegs servos
       int i;
       for (i=0;i<3;i++){
          dxl_write_word( whegs_sx[i], P_MOVING_SPEED_L, 0 );
          dxl_write_word( whegs_dx[i], P_MOVING_SPEED_L, 1024 );
          dxl_write_word( whegs_sx[i], 8, 0 );
          dxl_write_word( whegs_dx[i], 8, 0 );
       }

       //Disable Continuous Turn for Body Torsion Servo and Tail Servo
       dxl_write_word( 10, 8, 1023 );//Disable Continuous Turn Mode
       dxl_write_word( 10, P_TORQUE_ENABLE, 0 ); //Disable Torsion Servo
       dxl_write_word( 1, 8, 1023 );//Disable Continuous Turn Mode
       dxl_write_word( 1, P_TORQUE_ENABLE, 0 ); //Disable Tail Servo
       // Write moving speed
       //dxl_write_word( 6, P_MOVING_SPEED_L, 150 );
       //dxl_write_word( 8, P_MOVING_SPEED_L, 1174 );

       //Prepare servos data
       sdata[0].id = whegs_sx[0];
       sdata[1].id = whegs_sx[1];
       sdata[2].id = whegs_sx[2];
```

```
        sdata[3].id = whegs_dx[0];
        sdata[4].id = whegs_dx[1];
        sdata[5].id = whegs_dx[2];
        sdata[6].id = 15; //Distance Sensor Servo


        dxl_write_word(15, 8, 1023); //Position Mode for Sensor Servo

        //Button Init
        DDRE  = 0x0C;
        PORTE = 0xF0;

        //Prepare Sensor
        dxl_write_byte( 100, OBSTACLE_DETECTED_COMPARE, 1 );//High Dist Sens
        //dxl_write_byte(100,OBSTACLE_DETECTED_COMPARE_VALUE, 50) ;
        //50 is detection threshold

        adc_init();
        setupAcc();
        setupSharp();
        initTimer1();
        walking = 1;
        while(1){

           //Read and Communicate Sensors Data
           readAllSensors(&sensors);
           printAllSensors(sensors);

           // Set Speed According to Body Inclination
           dnspeed = (600 + 4 *sensors.body_inclination_xg); //set speed according to terrain inclination(more speed =
more force)

           //Main Behaviors Section
           //Vars for Behaviors
           float ld=abs((int)(sensors.distance_floor_left -sensors.distance_floor_center));
           float rd=abs((int)(sensors.distance_floor_right -sensors.distance_floor_center));

           //Jump Down Behavior
           if (sensors.abdomen_inclination_g < -45 && sensors.distance_front_center < 30){
        //..isFalling, TerrainIsNear -> JUMP!
                    stop();
                    comm_err = 0;                 //communication error
                    while(    sensors.abdomen_inclination_g < -45 &&
           sensors.distance_front_center < 30){//JUMP
                              //printf("\nJUMP\n");
                              if (comm_err == 0 )dpAbd = dpAbd + 1;
                              //LIFT BODY
                              comm_err = 0;//reset communication error
                              setAbdomenDeg(dpAbd, 1000);//Lift
                              if (checkDxlConn()==-1)
                                        comm_err = 1;
                              else{
                                        int isMoving = 1;
                                        while(isMoving == 1) {//reach pos
                                                  isMoving = dxl_read_byte(13,P_MOVING);
                                                  if (checkDxlConn()==-1){
                                                            comm_err = 1;
                                                            isMoving = 0; //permits exiting from while
                                                  }
                                        }
                              }
                              sensors.abdomen_inclination_g = getAbdomenDegG();
                              sensors.distance_front_center = readSharp_cm(4);
                    }
           }else
           //Terrain Check Behavior
           if(0&&  (ld > 8 || rd > 8)) { //terrain not uniform //last=4
                    //printf("ld:%d,rd:%d\n",(int)ld,(int)rd);
                    if (ld >= rd) {turnR = 1; turnL=0;}
                    if (ld < rd) {turnL = 1; turnR=0;}
           }else
           if (sensors.distance_front_center <  15){//Near Center Obstacle..
                    stop();//STOP WALKING
                    //Approaching Obstacle Behavior
                    comm_err = 0;                 //communication error
                    while(sensors.distance_front_center<15 && sensors.abdomen_inclination<80)
                    {//LIFT UP..
                              //printf("\nLIFT UP TO CLIMB\n");
                              if (comm_err == 0 )dpAbd = dpAbd + 1;
                              if (dpAbd > 80)dpAbd = 80;
```

```
                        //LIFT BODY
                        comm_err = 0;//reset communication error
                        setAbdomenDeg(dpAbd, 200);//Lift
                        if (checkDxlConn()==-1)
                                comm_err = 1;
                        else{
                                int isMoving = 1;
                                while(isMoving == 1) {//reach pos
                                        isMoving = dxl_read_byte(13,P_MOVING);
                                        if (checkDxlConn()==-1){
                                                comm_err = 1;
                                                isMoving = 0; //permits exiting from while
                                        }
                                }
                        }
                        sensors.distance_front_center = readSharp_cm(4);
                }
        }
        else //Adapt to Floor Behavior
        if(sensors.distance_floor_center > 23 ){ // Floor too distant ..
                stop();
                comm_err = 0;                   //communication error
                while(    sensors.distance_floor_center > 23 &&
                        sensors.abdomen_inclination > -75)
                {// LOWER DOWN..
                        //printf("\nLOWERING TO THE FLOOR\n");
                        if (comm_err == 0 )dpAbd = dpAbd - 1;
                        if (dpAbd < -75)dpAbd = -75;
                        comm_err = 0;//reset communication error
                        setAbdomenDeg(dpAbd, 200);//Lower Abdomen
                        if (checkDxlConn()==-1)
                                comm_err = 1;
                        else{
                                int isMoving = 1;
                                while(isMoving == 1) {//reach pos
                                        isMoving = dxl_read_byte(13,P_MOVING);
                                        if (checkDxlConn()==-1){
                                                comm_err = 1;
                                                isMoving = 0; //permits exiting from while
                                        }
                                }
                        }
                        sensors.distance_floor_center = readSharp_cm(6);
                }
        }
        //End of Behaviors
        //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 );

                                _delay_ms(40000);//Mantain behavior
                                //Disable behavior..
                                turnL = 0;
                                turnR = 0;

                        }

                }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 );
                                _delay_ms(40000);//Mantain behavior
                                //Disable behavior..
                                turnL = 0;
                                turnR = 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 );
```

```
                                        _delay_ms(40000);//Mantain behavior
                                        //Disable behavior..
                                        turnL = 0;
                                        turnR = 0;
                                }
                        }else
                        if (!(turnL + turnR)){//If not turning
                                go_fwd();//Restart walking
                        }
                }else{//Stop Walking
                        stop();
                }
                //Tail Control Section
                pTail = (unsigned short)dxl_read_word( 1, P_PRESENT_POSITION_L);
                //printf("%d ",pTail);

                //Tail Behaviors Manager
                // Check if desidered Tail position (if was set) has been reached
                if(dpTail != -1 && abs((int)dpTail - (int)pTail) < 50){
                        dxl_write_word( 1, P_TORQUE_ENABLE, 0 );
                        //This ^^^ disables Torsion Servo
                        // Needed to shut off tail behaviors
                        // activated in previous loop cycle
                        dpTail = -1; //disable desidered Tail position (any will be good)
                }

                //Tail Behavior 1: Avoiding Falling Backward
                //If tail is high and  abdomen is'n lowand body is tilted up of at least 45°
                if (parallel && pTail < 300 && getAbdomenDeg() >-10 && readAccDeg(1) > 45)
                {       //avoid falling backward
                        //printf("\navoid falling backward\n");
                        dpTail = 500;
                        dxl_write_word( 1, P_GOAL_POSITION_L, dpTail );
                        dxl_write_word( 1, P_MOVING_SPEED_L, 210 );
                }

                //Tail Behavior 2: Climbing
                //If tail is a slightly high and abdomen is slightly low and body is tilted up
                //of at least 60°
                if (parallel && pTail < 450 && getAbdomenDeg() < -20 && readAccDeg(1) > 60 )
                {       //climb
                        //printf("\nclimb\n");
                        dpTail = 712;
                        dxl_write_word( 1, P_GOAL_POSITION_L, dpTail );
                        dxl_write_word( 1, P_MOVING_SPEED_L, 210 );
                }else


                //Working on timer..
//              time = Timer1Read();
//              //printf(" %u ", time/(unsigned int)TIME_MILLISEC);
//              printf(" %u ", time);
                //printf(" main ");
                //printf("\n");

                if(0){//STOP MOTION
                        //printf("PRESS KEY\n");
                        torque(0);
                        unsigned char ReceivedData = getchar();
                        torque(1);
                }
        }  //main loop
        return 0;
}


void printAllSensors(Sensors sensors){
        //printf("dnspeed:\t%d\t",dnspeed);
        printf("FC %d\t",(int)(sensors.distance_front_center));
        printf("GC %d\t",(int)(sensors.distance_floor_center));
        printf("GL %d\t",(int)(sensors.distance_floor_left));
        printf("GR %d\t",(int)(sensors.distance_floor_right));
        printf("Z %d\t",(short)(1000*readAccNorm(0)));
        printf("X %d\t",(short)(1000*readAccNorm(1)));
        printf("Y %d\t",(short)(1000*readAccNorm(2)));
        printf("AD %d\t",getAbdomenDeg());
        printf("TD %d\n",getTailDeg());
}
```

```
// Print communication result
void PrintCommStatus(int CommStatus)
{
	switch(CommStatus)
	{
	case COMM_TXFAIL:
	   printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
	   break;

	case COMM_TXERROR:
	   printf("COMM_TXERROR: Incorrect instruction packet!\n");
	   break;

	case COMM_RXFAIL:
	   printf("COMM_RXFAIL: Failed get status packet from device!\n");
	   break;

	case COMM_RXWAITING:
	   printf("COMM_RXWAITING: Now recieving status packet!\n");
	   break;

	case COMM_RXTIMEOUT:
	   printf("COMM_RXTIMEOUT: There is no status packet!\n");
	   break;

	case COMM_RXCORRUPT:
	   printf("COMM_RXCORRUPT: Incorrect status packet!\n");
	   break;

	default:
	   printf("This is unknown error code!\n");
	   break;
	}
}

// Print error bit of status packet
void PrintErrorCode()
{
	if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
	   printf("Input voltage error!\n");

	if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
	   printf("Angle limit error!\n");

	if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
	   printf("Overheat error!\n");

	if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
	   printf("Out of range error!\n");

	if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
	   printf("Checksum error!\n");

	if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
	   printf("Overload error!\n");

	if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
	   printf("Instruction code error!\n");
}

unsigned int Timer1Read( void )
{
unsigned char sreg;
unsigned int i;
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
cli() ;//__disable_interrupt();
/* Read TCNTn into i */
i = TCNT1;
/* Restore global interrupt flag */
SREG = sreg;
return i;
}


void Timer1Write( unsigned int i )
{
unsigned char sreg;
```

```
/* Save global interrupt flag */
sreg = SREG;
/* Disable interrupts */
cli();//__disable_interrupt();
/* Set TCNTn to i */
TCNT1 = i;
/* Restore global interrupt flag */
SREG = sreg;
}

void Timer1Pause(unsigned int flag){/*Pause using timer 1 */
       Timer1Write(0);
       unsigned int time = Timer1Read();
       while(time < flag ){
          time = Timer1Read();
       }
}

void initTimer1(void){
       TCCR1B = (1<<CS12)|(1<<CS10);//Prescaler to Sys Clock / 1024 = (16MHz / 1024)
}

void init_OC1A(void)
{
//ASSR= 1<<AS2; // Enable asynchronous mode
TCCR1B = (1<<CS12)|(1<<CS10);//Prescaler to Sys Clock / 1024 = (16MHz / 1024)
TIFR1 = 1<<OCF1A; // Clear OCF1/ Clear pending interrupts
TIMSK1= 1<<OCIE1A; // 1:Enable (0:Disable) Timer1 Output Compare Mode Compare Match Interrupt
OCR1A= 48; // Set Output Compare Value to 32
//Timer1Write((unsigned int) 0);//Resets Timer
//DDRB= 0xFF; // Set Port B as output
//while (ASSR&(1<<OCR2UB)); // Wait for registers to update
}

//Interrupt Function


ISR(TIMER1_COMPA_vect){
       // Clear timer on compare match (now manually)
       Timer1Write((unsigned int) 0);//Resets Timer
       float deltaT = 1000/dnspeed; //msecs per count (AX12 has 1023 counts for 300 deg)
       OCR1A = TIME_MILLISEC * deltaT;// set compare interval
}


unsigned short mirror(unsigned short speed){//calc specular speed
       if (speed > 1023)
          return speed - 1024;
       else
          return speed + 1024;
}

void adc_init(void) // Function used to initialise the ADC feature
{
       ADMUX=0X40; // 0x40 for 10bit //0x60 for 8bit
       ADCSRA=0X87; // We have set the ADSC bit to start a conversion, and the
       // ADPS bits are set so that the prescaler is 128
       //ADCSRA=0X80; // ADEN is set, to enable the ADC
}

uint16_t adc_start(unsigned char channel)// Function to perform an ADC conversion, Takes 0-8 as input
                              // to select which input to convert
       {
       unsigned char i;

       i=channel&0x07; // Decide which line to perform ADC conversion on
       //ADMUX=i|0x60; // Enter which line to perform in the ADC control register (for 8 bit)
       ADMUX=i|0x40; // Enter which line to perform in the ADC control register (for 10 bit)
       ADCSRA|=1<<ADSC;

       while(ADCSRA & (1<<ADSC));  // wait for conv. to complete
       //unsigned char temp=ADCH; //for 8-bit
       unsigned short temp;
       temp=ADC;//for 10-bit
       return temp;
}

int checkDxlConn(void){
       CommStatus = dxl_get_result();
       if( CommStatus != COMM_RXSUCCESS ){
```

```
            PrintCommStatus (CommStatus);
            PrintErrorCode();
             return -1;
        }
        return 1;
}

unsigned short mirrorPos(unsigned short pos){/* Calc pos for upside-down servo */
        pos = 1023 - pos;
        return pos;
}

unsigned short pos2deg(unsigned short pos){ /* deg : 0 to 300 . 150 is center */
        if(pos > 1023)return -1; /* Error: out of range */
        //printf(" p%u ",pos);
        uint32_t deg;
        //unsigned int deg;
        deg = (uint32_t)pos;
        //printf(" d%u ",deg);
        deg = deg * 293; //it was 29.3
        //printf(" d%u ",deg);
        deg = deg / 1000;
        //printf(" d%u ",deg);
        return (unsigned short)deg;

}
unsigned short deg2pos(unsigned short deg){/* deg : 0 to 300 . 150 is center */
        if (deg > 300) return -1 ; /* Error: out of range */
        //printf(" p%u ",deg);
        uint32_t pos;
        //unsigned int deg;
        pos = (uint32_t)deg;
        //printf(" d%u ",pos);
        pos = pos * 1000;
        //printf(" d%u ",pos);
        pos = pos / 293;//it was 29.3
        //printf(" d%u ",pos);
        return (unsigned short)pos;

}

short  getTailDeg(void){
        short deg = (short)pos2deg(dxl_read_word(1, P_PRESENT_POSITION_L)) - 150;
        return  -deg; //servo is mounted upside-down
}
void setTailDeg(short deg, unsigned short speed){
        deg = - deg; //servo is mounted upside-down
        if (deg < -90 || deg > 90 ){/* Error: out of range */
           printf("\nError: deg must be in the -90 to 90 deg range.\n");
        } else{
           unsigned short posTail = -1;//holds abdomen joint position
           posTail = deg2pos((unsigned short)(deg + 150));
           //printf ("1pos:\t%d\t",posAbdomen);
           dxl_write_word(1, P_MOVING_SPEED_L, speed);
           dxl_write_word(1, P_GOAL_POSITION_L, posTail);
        }
}

short  getAbdomenDeg(void){
        return (short)pos2deg(dxl_read_word(13, P_PRESENT_POSITION_L)) - 150;
}

short  getAbdomenDegG(void){ /* Gets inclination wrt g-plane */
        return getAbdomenDeg() + (short)readAccDeg(1);
}


void setAbdomenDeg(short deg, unsigned short speed){
        if (deg < -75 || deg > 80 ){/* Error: out of range */
           printf("\nError: deg must be in the -75 to + 80 deg range.\n");
        } else{
           unsigned short posAbdomen = -1;//holds abdomen joint position
           posAbdomen = deg2pos((unsigned short)(deg + 150));
           //printf ("13pos:\t%d\t",posAbdomen);
           dxl_write_word(13, P_MOVING_SPEED_L, speed);
           dxl_write_word(13, P_GOAL_POSITION_L, posAbdomen);
           posAbdomen = mirrorPos(posAbdomen ) ;
           //printf ("19pos:\t%d\n",posAbdomen);
           dxl_write_word(19, P_MOVING_SPEED_L, speed);
           dxl_write_word(19, P_GOAL_POSITION_L, posAbdomen);
```

```
        }
}

unsigned short dist2cm(unsigned short dist){//Convert distance from SHARP GP2Y0A21YK sensor to cm
        uint32_t cm = (uint32_t)dist;
        cm = 12344 * cm ;
        return cm;
}



//
// Converting the values of the IR distance sensor to centimeters
// Returns -1, if the conversion did not succeed
//
float ir_distance_calculate_cm(ir_distance_sensor sensor, unsigned short adc_value){
        if (adc_value + sensor.b <= 0)
        {
           return -1;
        }
        float dist_cm;
        dist_cm = (float)sensor.a / (float)(adc_value + sensor.b) - (float)sensor.k;
        //dist_cm =  5461 / (adc_value + -17) - 2;
        //printf("cm%d ",dist_cm);
        return dist_cm;
}

float cosML(float x){ //cosine MacLaurin 4 term  (GOOD from 0∞ to 45∞ ,need to improve to 50∞)
        //deg 2 rad
        x = x * 0.017453;
        //printf("r%d ",(int)(x*10));
        //cos(x):  1-0.5x^2 +  0.04167 * x^4
        float cosine;
        cosine = 1- 0.5 * x*x + 0.04167 * x *x *x *x ;
        //printf("cos%d ",(int)(cosine*10));
        return cosine;
}

float readHeight(void){
        //Read Sensor
        uint16_t result=adc_start(6);    // cHANNEL 6
        //printf("r%u ", result);
        float dist_cm;
        dist_cm = ir_distance_calculate_cm(GP2Y0A21YK, result);//Convert
        //printf("cm%d ", (int)(dist_cm*10));
        float height ;
        height = 4.5 +  (float)dist_cm;//Add sensor stand length to Height
        if(height<10) height = 10;
        else if(height>80) height = 80;
        return height ;
}

void go_fwd(void){
        int i;
        for (i=0;i<3;i++)
           dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, dnspeed );
        for (i=3;i<6;i++)
           dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, mirror(dnspeed ));
}

void stop(void){
        int i ;
        for (i=0;i<3;i++)
                    dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, 0 );
        for (i=3;i<6;i++)
                    dxl_write_word( sdata[i].id, P_MOVING_SPEED_L, mirror(0));
}

void torque(unsigned char val){
        int i ;
        for (i=0;i<3;i++)
           dxl_write_word( sdata[i].id, P_TORQUE_ENABLE, val );
        for (i=3;i<6;i++)
           dxl_write_word( sdata[i].id, P_TORQUE_ENABLE, val);
}



unsigned char readAXS1cd(void){//Read AX-S1 Sensor Center Distance Filtering Errors and Compensating Ambient Light
        int i,dCIR,lCIR,cdCIR,dCIRmin;//corrected distance
```

```
	dCIRmin = 255;
	for(i=0;i<10;i++){//Collect min val (avoid ambient ir-modulated interferences )
	   do{	dCIR = (unsigned char) dxl_read_byte( 100, CENT_IR_SENSOR_DATA);}
		while (dxl_get_result()!=COMM_RXSUCCESS);
	   if (dCIR < dCIRmin) dCIRmin = dCIR;
	}//using dCIRmin as distance value
	do{lCIR = (unsigned char) dxl_read_byte(100,CENTER_LUMINOSITY);}
	   while (dxl_get_result()!=COMM_RXSUCCESS);
	cdCIR = dCIRmin - 2*lCIR ;//Compensate Ambient Luminosity
	if(cdCIR <0)cdCIR = 0;//Lower bound
	//printf("dCIRmin %d \t",dCIRmin);
	//printf("lCIR %d \t",lCIR);
	//printf("cdCIR %d \n",cdCIR);
	//if(dCIRmin > 100)printf(" >100 \n");
	return cdCIR;
}


void setupAcc(void){/* Configure MCU bits to use Accelerometer */
	//Configure MCU outputs to interface Acc inputs
	DDRA |= (1<<PA7)|(1<<PA6) ;//MCU PIN A6 and A7 set as Output
}

uint16_t readAcc(unsigned char addr){
	/* Note: All PORTA pin values are inverted (1 is LOW, 0 is HIGH)
	 * This is due to cm-510 output transistor, I guess
	 */
	switch (addr){
	case 0 :
	   // Choosing Z Accel.
	   PORTA = (1<<PA7)|(1<<PA6); //MPLEX: ADDR 00 (0)
	   //printf("\n00:Z \t");
	   break;

	case 1 :
	   // Choosing X Accel.
	   PORTA = (1<<PA7)|(0<<PA6); //MPLEX: ADDR 01 (1)
	   //printf("\n01:X \t");
	   break;

	case 2 :
	   // Choosing Y Accel.
	   PORTA = (0<<PA7)|(1<<PA6); //MPLEX: ADDR 10 (2)
	   //printf("\n10:Y\t");
	   break;

	case 3 :
	   // Chosing Free Address
	   PORTA = (0<<PA7)|(0<<PA6); //MPLEX: ADDR 11 (3)
	   //printf("\n11:F\t");
	   break;
	default:
		printf("\nWrong Address (%d) Selected on channel 2\n", addr);
		return -1;
	}

	//Pause before acquiring..
	Timer1Write(0);
	int time = Timer1Read();
	while(time < TIME_MILLISEC){
	   time = Timer1Read();
	   //printf("time=%d\n",time);
	}
	int16_t result=adc_start(2);	//Channel 2
	//printf("result : \t%d\n",result);
	return result;
}

void setupSharp(void){/* Configure MCU bits to use Sharp IR Sensors Board */
	//Configure MCU outputs to interface Board inputs
	DDRA |= (1<<PA2)|(1<<PA3);//MCU PIN PA2 and PA3 set as Output
	DDRF |= (1<<PF4) ;//MCU PIN PF4 set as Output
}

uint16_t readSharp(unsigned char addr){ /* Read value of Sharp sensor mapped on address "addr" */
	/* Note: All PORTA pin values are inverted (1 is LOW, 0 is HIGH)
	 * This is due to cm-510 output transistor, I guess
	 */
	switch(addr){
	   case 0:   //MPLEX: ADDR 000 (0)
```

```
                    PORTA = (1<<PA3)|(1<<PA2); //BIT1:0 = 00
                    PORTF = (0<<PF5);//BIT2 = 0
                    break;
            case 1:   //MPLEX: ADDR 001 (1)
                    PORTA = (1<<PA3)|(0<<PA2); //BIT1:0 = 01
                    PORTF = (0<<PF5);//BIT2 = 0
                    break;
            case 2:   //MPLEX: ADDR 010 (2)
                    PORTA = (0<<PA3)|(1<<PA2); //BIT1:0 = 10
                    PORTF = (0<<PF5);//BIT2 = 0
                    break;
            case 3:   //MPLEX: ADDR 011 (3)
                    PORTA = (0<<PA3)|(0<<PA2); //BIT1:0 = 11
                    PORTF = (0<<PF5);//BIT2 = 0
                    break;
            case 4:   //MPLEX: ADDR 100 (4)
                    PORTA = (1<<PA3)|(1<<PA2); //BIT1:0 = 00
                    PORTF = (1<<PF5);//BIT2 = 1
                    break;
            case 5:   //MPLEX: ADDR 101 (5)
                    PORTA = (1<<PA3)|(0<<PA2); //BIT1:0 = 01
                    PORTF = (1<<PF5);//BIT2 = 1
                    break;
            case 6:   //MPLEX: ADDR 110 (6)
                    PORTA = (0<<PA3)|(1<<PA2); //BIT1:0 = 10
                    PORTF = (1<<PF5);//BIT2 = 1
                    break;
            case 7:   //MPLEX: ADDR 111 (7)
                    PORTA = (0<<PA3)|(0<<PA2); //BIT1:0 = 11
                    PORTF = (1<<PF5);//BIT2 = 1
                    break;
            default :
                    printf("\nWrong Address (%d) Selected on channel 2\n", addr);
                    return -1;
        }
        //Pause before acquiring..
        Timer1Write(0);
        int time = Timer1Read();
        while(time < TIME_MILLISEC ){
            time = Timer1Read();
            //printf("time=%d\n",time);
        }
        int16_t result = adc_start(6);   //Channel 6
        //filtering : pass only 0.3V (64) - 3V (613)
        if (result < 64)
            result = 64;
        if (result > 613)
            result = 613;
        //printf("result : \t%d\n",result);
        return result;
}

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}
};

float readSharp_cm(unsigned char addr){ /* Read value of Sharp sensor mapped on address "addr"  and convert to cm
distance */
        float val_cm;
        unsigned short val_10bit = readSharp(addr);
        int i;
        for(i = 0; i<17; i++){
            if (val_10bit >=  sharp_calib[i][0] && val_10bit < sharp_calib[i+1][0]){
                    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]));
                    return val_cm;
            }
        }
        return -1; /* Error, data is out of range */
}

float readAccNorm(unsigned char addr){/* read calibrated and normalized (0-1) value from Accel Channel @ addr */
        float norm_val = 0;
        if (addr == 1){ /* Calibr for X */
            norm_val = (float)((float)readAcc(addr) - (float)325)/(float)164; //325 : 0-val , 164 : mod(g)
        }else
        if (addr == 2){ /* Calibr for Y */
            norm_val =  (float)((float)readAcc(addr) - (float)338)/(float)162; //339 : 0-val , 164 : mod(g)
        }else
        if (addr == 0){ /* Calibr for Z */
            norm_val =  (float)((float)readAcc(addr) - (float)286)/(float)164; //286 : 0-val , 164 : mod(g)
        }
        else{/* Wrong Channel (addr) selected */
            return -1;
        }
        /* If you are here then the channel was valid .. */
        //Perform value bounding..
        if (norm_val > 1) norm_val = 1;
        else if (norm_val < -1) norm_val = -1;
        return norm_val;

}

float readAccDeg(unsigned char addr){/* read deg value from Accel on channel @ addr */
        float val = readAccNorm(addr);
        //printf("\n\tnormval=%d\n", (int)(1000*val));
        val =        asin(val);
        //printf("\n\tasinval=%d\n", (int)(1000*val));
        val = val * 57.296739;//rad2deg
        return val;
}


void readAllSensors(Sensors *s){
        s->distance_front_center = readSharp_cm(4);
        s->distance_floor_center = readSharp_cm(6);
        s->distance_floor_left = readSharp_cm(5);
        s->distance_floor_right = readSharp_cm(7);
        s->body_inclination_xg = readAccDeg(1);
        s->body_inclination_yg = readAccDeg(2);
        s->body_inclination_zg = readAccDeg(0);
        s->abdomen_inclination = getAbdomenDeg();
        s->abdomen_inclination_g = getAbdomenDegG();
}

int facingAbyss(void){
        //TODO
        return -1;
}
```

Appendix C : Software

In this section there are:

- the source code of the three plugins for the Player server.
- The **configuration files** needed to use plugins

Source Code

C++ Code
Text 2: Position Plugin Code

Plugin: *LionHell_pos.so*

Filename: ***LionHell_pos.h***

```
#include <libplayercore/playercore.h>

class LionHell_pos : public ThreadedDriver{
       private:
          player_position3d_data_t data , prev_data ;

          //Serial data
          int fx, fy, fz; //Forces/g_force ratios along x,y,z axes [adimensional]

          /* FUNCTIONS */

          //Publish state data .
          void PutData ( void ) ;

       public:
          LionHell_pos(ConfigFile *cf, int section);
          ~LionHell_pos(void);

          //Thread Life-Cycle
          virtual void Main();
          virtual int MainSetup();
          virtual void MainQuit();

          //Message Handling
          virtual int ProcessMessage(MessageQueue* queue, player_msghdr* msghdr, void* data);
};
```

Filename: ***LionHell_pos.cpp***

```
#include <LionHell_pos.h>
#include <iostream>
int LionHell_openSerial(void);
int LionHell_configSerial(int fd);
static int LionHell_getSerial(int *fx, int *fy, int* fz );


LionHell_pos::LionHell_pos(ConfigFile *cf, int section)
       : ThreadedDriver(cf, section, false, PLAYER_MSGQUEUE_DEFAULT_MAXLEN,
                        PLAYER_POSITION3D_CODE)
{
       /* Init data */
       memset(&data, 0, sizeof(data));
       data.pos.px = 0;
       data.pos.py = 0;
       data.pos.pz = 0;
       data.pos.proll = 0;
       data.pos.ppitch = 0;
       data.pos.pyaw = 0;
       data.vel.px = 0;
       data.vel.py = 0;
       data.vel.pz = 0;
```

```
	data.vel.proll = 0;
	data.vel.ppitch = 0;
	data.vel.pyaw = 0;
	data.stall = 0;
	std::cout<<std::endl<<"LionHell_pos instance Constructed"<<std::endl;
}

LionHell_pos:: ~LionHell_pos(void)
{
	std::cout<<std::endl<<"LionHell_pos instance Destructed"<<std::endl;
}

int LionHell_pos::MainSetup(){
	std::cout<<std::endl<<"Setup"<<std::endl;
	return 0 ;
}

void LionHell_pos::MainQuit(){
	std::cout<<std::endl<<"Shutdown"<<std::endl;
}

void LionHell_pos::Main(void){
	while(1){
	   pthread_testcancel();
	   std::cout<<std::endl<<"Main"<<std::endl;

	   if (LionHell_getSerial(&fx, &fy, &fz) == 9) {
		    /* Do something with data *
		     * Accelerometer alone is not sufficient
		     * to calc pos,vel data.
		     * We need other sensors to know
		     * orientation wrt gravity vector.
		     *
		     * ACCELEROMETER EQUATIONS:
		     * fx = gx + fmx
		     * fy = gy + fmy
		     * fz = gz + fmz
		     *
		     * Notation:
		     * fx:  force from accelerometer
		     * gx:  force due to gravity
		     * fmx: force due to movement
		     * */
	   }
	   ProcessMessages(); //process incoming messages
	   PutData();
	   usleep(10000);
	}
}

void LionHell_pos::PutData ( void ){
	std::cout<<std::endl<<"PutData"<<std::endl;
	Publish(device_addr, PLAYER_MSGTYPE_DATA, PLAYER_POSITION3D_DATA_STATE, (void *) &data, sizeof(data)) ;
	std::cout<<std::endl<<"Publish done"<<std::endl;
}

int LionHell_pos::ProcessMessage(MessageQueue* queue, player_msghdr* msghdr, void* data){
	std::cout<<std::endl<<"ProcessMessage"<<std::endl;
	//PLAYER_WARN("unknown_message");
	//return -1;
	return 0;
}

Driver* LionHell_pos_Init(ConfigFile* cf, int section){
	std::cout<<std::endl<<"LionHell_pos - Init"<<std::endl;
	return (Driver*) (new LionHell_pos(cf, section));
}

void LionHell_pos_Register(DriverTable* table)
{
 table->AddDriver("LionHell_pos", LionHell_pos_Init ) ;
}

extern "C" {
	int player_driver_init(DriverTable* table){
	   PLAYER_MSG0 (1, "Registering LionHell_pos driver.");
	   LionHell_pos_Register(table);
	   return 0 ;
	}
}
```

```
#include <fcntl.h> // File control definitions
#include <termios.h> // POSIX terminal control definitionss
static int LionHell_getSerial(int *fx, int *fy, int* fz){
       //momentarily here only to read scanf
       int i; //to be ignored
    char buf[BUFSIZ];
    int fd, ret;
       fd =  LionHell_openSerial();
       LionHell_configSerial(fd);

    ret = read(fd, buf, sizeof(buf));  /* <--  IL PROBLEMA E' QUI ! */
       printf("\nret = %d\n", ret);
       printf("\nbuf = %s\n", buf);
    if (ret > 0) {
          ret = 0;
          ret = sscanf(buf, "FC %d\tGC %d\tGL %d\tGR %d\tZ %d\tX %d\tY %d\tAD %d\tTD %d\r\n", &i, &i, &i, &i, fz , fx,
fy, &i, &i);
          printf("\nsscanf: ret = %d\n", ret);
    }else if (ret < 0) {
          PLAYER_ERROR1("error reading: %s", strerror(errno));

    }else {
       PLAYER_ERROR("Failed to read position");
       ret = 1;
    }

    if (close (fd))
      PLAYER_ERROR1("error closing: %s", strerror(errno));

    return ret;
}

int LionHell_openSerial(void){
       int fd; // file description for the serial port

       fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);

       if(fd == -1) // if open is unsucessful
       {
          //perror("open_port: Unable to open /dev/ttyS0 - ");
          printf("open_port: Unable to open /dev/ttyUSB0. \n");
       }
       else
       {
          fcntl(fd, F_SETFL, 0);
          printf("port is open.\n");
       }

       return(fd);
} //open_port

int LionHell_configSerial(int fd)      // configure the port
{
       struct termios port_settings;      // structure to store the port settings in
       memset(&port_settings,0,sizeof(port_settings));
       cfsetispeed(&port_settings, B57600);    // set baud rates
       cfsetospeed(&port_settings, B57600);

       port_settings.c_iflag=0;
       port_settings.c_oflag=0;
       port_settings.c_cflag=CS8|CREAD|CLOCAL;           // 8n1, see termios.h for more information
       port_settings.c_lflag=ICANON;//original: 0 , others: ICANON
       port_settings.c_cc[VMIN]=255; //was 1 //line length is 75
       port_settings.c_cc[VTIME]=5;

       tcsetattr(fd, TCSANOW, &port_settings);    // apply the settings to the port
       return(fd);


} //configure_port
```

Plugin: *LionHell_ranger.so*

C++ Code

Text 3: Ranger Plugin Code

Filename: ***LionHell_ranger.h***

```
#include <libplayercore/playercore.h>

class LionHell_ranger : public ThreadedDriver{
       private:
          player_ranger_data_range_t data;
          player_ranger_geom_t geom;

          //Serial data
          double fc, gc, gl, gr; //Distances [cm]
          int ad;//abdomen angle. positive is up [deg]

          //Common geometry data
          double PI; //PI : PIGRECO. Used in deg2rad conversion.
          double servo_center_x;//[m]

          //Following Sensors  Geometry Data are wrt robot center
          //LionHell center is ..need a picture?

          //Front Center Sensor geometry data
          double servo_link_length_to_fc;//[m]
          double servo_link_angle_offset_to_fc; //[deg]
          double fc_pose_y ; //[m]
          double fc_deg_x ; //[deg]
          double fc_deg_z ; //[deg]
          double fc_pose_x ; //[m]
          double fc_pose_z ; //[m]
          double fc_deg_y;//[deg]

          /* FUNCTIONS */
          void updateRanges(void);
          void updateGeometry(void);

          //Publish state data .
          void PutData ( void ) ;

       public:
          LionHell_ranger(ConfigFile *cf, int section);
          ~LionHell_ranger(void);

          //Thread Life-Cycle
          virtual void Main();
          virtual int MainSetup();
          virtual void MainQuit();

          //Message Handling
          virtual int ProcessMessage(MessageQueue* queue, player_msghdr* msghdr, void* data);
};
```

Filename: ***LionHell_ranger.cpp***

```
#include "LionHell_ranger.h"
#include <iostream>
#include <math.h>
int LionHell_openSerial(void);
int LionHell_configSerial(int fd);
static int LionHell_getSerial(double* fc, double* gc, double* gl, double* gr, int* ad);


LionHell_ranger::LionHell_ranger(ConfigFile *cf, int section)
       : ThreadedDriver(cf, section, false, PLAYER_MSGQUEUE_DEFAULT_MAXLEN,
                     PLAYER_RANGER_CODE)
{
       memset(&data, 0, sizeof(data));
       data.ranges_count = 4;
```

```
	data.ranges = new double[data.ranges_count];
	for(unsigned int i = 0; i < data.ranges_count; i++){
	   data.ranges[i] = -1;
	}
	memset(&geom, 0, sizeof(geom));

	//Following Sensors  Geometry Data are wrt robot center
	//LionHell center is ..need a picture?

	/* Geometry data setup */
	PI = 3.14159265;

	/* Set sensor's size */
	geom.size.sw = 0.015; //along x axis
	geom.size.sl = 0.045; //along y axis
	geom.size.sh = 0.013; //along z axis

	/* Set sensors geom. */
	/* Common geometry data */
	servo_center_x = 0.143 ;//[m]

	/* Front Center Servo */
	/* Fixed Data */
	servo_link_length_to_fc = 0.155 ;//[m]
	servo_link_angle_offset_to_fc = 35 ; //[deg]
	fc_pose_y = 0; //[m]
	fc_deg_x = 0; //[deg]
	fc_deg_z = 0; //[deg]
	/* Variable data : will be updated by getSerial(..) */
	/* Suppose Abdomen joint angle is 0 deg.*/
	fc_pose_x = servo_center_x + servo_link_length_to_fc; //[m]
	fc_pose_z = 0; //[m]
	fc_deg_y = 0; //[deg]

	std::cout<<std::endl<<"LionHell_ranger instance Constructed"<<std::endl;
}

LionHell_ranger:: ~LionHell_ranger(void)
{
	delete[] data.ranges;
	std::cout<<std::endl<<"LionHell_ranger instance Destructed"<<std::endl;
}

int LionHell_ranger::MainSetup(){
	std::cout<<std::endl<<"Setup"<<std::endl;
	return 0 ;
}

void LionHell_ranger::MainQuit(){
	std::cout<<std::endl<<"Shutdown"<<std::endl;
}

void LionHell_ranger::Main(void){
	while(1){
	   pthread_testcancel();
	   std::cout<<std::endl<<"Main"<<std::endl;

	   if (LionHell_getSerial(&fc, &gc, &gl, &gr, &ad) == 0) {
	            printf("\ngetSerial SUCCEDEED\n");
	            printf("\n pose.px: %g\n", geom.pose.px);
	   }
	   updateRanges();
	   updateGeometry();
	   ProcessMessages(); //process incoming messages
	   PutData();
	   usleep(10000);
	}
}

void LionHell_ranger::PutData ( void ){
	std::cout<<std::endl<<"PutData"<<std::endl;
	Publish(device_addr, PLAYER_MSGTYPE_DATA, PLAYER_RANGER_DATA_RANGE, (void *) &data, sizeof(data)) ;
	Publish(device_addr, PLAYER_MSGTYPE_DATA, PLAYER_RANGER_DATA_GEOM, (void *) &geom, sizeof(geom)) ;
	std::cout<<std::endl<<"Publish done"<<std::endl;
}

int LionHell_ranger::ProcessMessage(MessageQueue* queue, player_msghdr* msghdr, void* data){
	std::cout<<std::endl<<"ProcessMessage"<<std::endl;
	//PLAYER_WARN("unknown_message");
	//return -1;
```

```
	return 0;
}

Driver* LionHell_ranger_Init(ConfigFile* cf, int section){
	std::cout<<std::endl<<"LionHell_ranger - Init"<<std::endl;
	return (Driver*) (new LionHell_ranger(cf, section));
}

void LionHell_ranger_Register(DriverTable* table)
{
 table->AddDriver("LionHell_ranger", LionHell_ranger_Init ) ;
}

extern "C" {
	int player_driver_init(DriverTable* table){
	   PLAYER_MSG0 (1, "Registering LionHell_ranger driver.");
	   LionHell_ranger_Register(table);
	   return 0 ;
	}
}

void LionHell_ranger::updateGeometry(void){
	if (ad != -1){
	   //Update Front Center Sensor Pose data
	   fc_pose_x = servo_center_x + servo_link_length_to_fc * cos((ad + servo_link_angle_offset_to_fc) * PI/180 );
//[m]
	   fc_pose_z = servo_link_length_to_fc * sin((ad + servo_link_angle_offset_to_fc) * PI/180 ); //[m]

	   fc_deg_y = ad;

	   /*
	   printf("\nfc_pose_x = %g\n", fc_pose_x);
	   printf("\nfc_pose_y = %g\n", fc_pose_y);
	   printf("\nfc_pose_z = %g\n", fc_pose_z);
	   */
	   geom.pose.px = fc_pose_x;
	   geom.pose.py = fc_pose_y;
	   geom.pose.pz = fc_pose_z;
	   geom.pose.proll = fc_deg_x;
	   geom.pose.ppitch = fc_deg_y;
	   geom.pose.pyaw = fc_deg_z;
	}
}

void LionHell_ranger::updateRanges(void){
	data.ranges[0] = fc;
	data.ranges[1] = gc;
	data.ranges[2] = gl;
	data.ranges[3] = gr;
}


//#include <stdio.h>
//#include <string.h> // string function definitions
//#include <unistd.h> // UNIX standard function definitions
#include <fcntl.h> // File control definitions
//#include <errno.h> // Error number definitions
#include <termios.h> // POSIX terminal control definitionss
//#include <time.h>
static int LionHell_getSerial(double* fc, double* gc, double* gl, double* gr, int* ad){
	//momentarily here only to read scanf
	int i; //to be ignored by sscanf

    char buf[BUFSIZ];
    int fd, ret;
	fd =  LionHell_openSerial();
	LionHell_configSerial(fd);
    ret = read(fd, buf, sizeof(buf));
	//printf("\nret = %d\n", ret);
	//printf("\nbuf = %s\n", buf);
    if (ret > 0){
	    ret = 0;
	    int sscanf_ret;
	    sscanf_ret = sscanf(buf, "FC %lf\tGC %lf\tGL %lf\tGR %lf\tZ %d\tX %d\tY %d\tAD %d\tTD %d\r\n", fc, gc, gl,
gr, &i , &i, &i, ad, &i);
	    printf("\nsscanf_ret = %d\n", sscanf_ret);
    } else if (ret < 0) {
	 PLAYER_ERROR1("error reading: %s", strerror(errno));

    } else {
```

```
        PLAYER_ERROR("Failed to read position");
        ret = 1;
    }

    if (close (fd))
      PLAYER_ERROR1("error closing: %s", strerror(errno));

    return ret;
}

int LionHell_openSerial(void){
       int fd; // file description for the serial port

       fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);

       if(fd == -1) // if open is unsucessful
       {
          //perror("open_port: Unable to open /dev/ttyS0 - ");
          printf("open_port: Unable to open /dev/ttyUSB0. \n");
       }
       else
       {
          fcntl(fd, F_SETFL, 0);
          printf("port is open.\n");
       }

       return(fd);
} //open_port

int LionHell_configSerial(int fd)      // configure the port
{
       struct termios port_settings;      // structure to store the port settings in
       memset(&port_settings,0,sizeof(port_settings));
       cfsetispeed(&port_settings, B57600);    // set baud rates
       cfsetospeed(&port_settings, B57600);
/*
       port_settings.c_cflag &= ~PARENB;    // set no parity, stop bits, data bits
       port_settings.c_cflag &= ~CSTOPB;
       port_settings.c_cflag &= ~CSIZE;
       port_settings.c_cflag |= CS8;
*/

       port_settings.c_iflag=0;
       port_settings.c_oflag=0;
       port_settings.c_cflag=CS8|CREAD|CLOCAL;           // 8n1, see termios.h for more information
       port_settings.c_lflag=ICANON;//original: 0 , others: ICANON
       port_settings.c_cc[VMIN]=255; //was 1 //line length is 75
       port_settings.c_cc[VTIME]=5;

       tcsetattr(fd, TCSANOW, &port_settings);    // apply the settings to the port
       return(fd);


} //configure_port
```

Plugin: *LionHell_robot.so*

C++ Code

Text 4: Robot Plugin Code

Filename: ***LionHell_robot.h***

```
#include <libplayercore/playercore.h>

class LionHell_robot : public ThreadedDriver{
       private:
          // My position3d interface
          player_devaddr_t my_position3d_addr;
          // My ranger interface
          player_devaddr_t my_ranger_addr;

          // Address of and pointer to the position3d device to which I'll subscribe
          player_devaddr_t position3d_addr;
```

```
        Device* position3d_dev;

        // Address of and pointer to the ranger device to which I'll subscribe
        player_devaddr_t ranger_addr;
        Device* ranger_dev;

        //My Ranger Interface data
        player_ranger_data_range_t ranger_data_range; //ranger ranges
        player_ranger_geom_t ranger_data_geom; //ranger geom
        player_position3d_data_t position3d_data_state; //position3d state

        /* FUNCTIONS */

        //Publish state data .
        void PutData ( void ) ;
        //Update my data
        void update_position3d_state(void* data);
        void update_ranger_range(void* data);
        void update_ranger_geom(void* data);

    public:
        LionHell_robot(ConfigFile *cf, int section);
        ~LionHell_robot(void);

        //Thread Life-Cycle
        virtual void Main();
        virtual int MainSetup();
        virtual void MainQuit();

        //Message Handling

        virtual int ProcessMessage(QueuePointer & resp_queue,
                                   player_msghdr * hdr,
                                   void * data);

        //virtual int ProcessMessage(MessageQueue* queue, player_msghdr* msghdr, void* data);
};
```

Filename: LionHell_robot.cpp

```
#include <LionHell_robot.h>
#include <iostream>

LionHell_robot::LionHell_robot(ConfigFile *cf, int section)
    : ThreadedDriver(cf, section)
{

    // Create my position3d interface
    if (cf->ReadDeviceAddr(&(this->my_position3d_addr), section,
                           "provides", PLAYER_POSITION3D_CODE, -1, NULL) != 0){
        this->SetError(-1);
        return;
    }
    if (this->AddInterface(this->my_position3d_addr)){
        this->SetError(-1);
        return;
    }


    // Create my ranger interface
    if (cf->ReadDeviceAddr(&(this->my_ranger_addr), section,
                                                    "provides", PLAYER_RANGER_CODE, -1, NULL) != 0){
        this->SetError(-1);
        return;
    }
    if (this->AddInterface(this->my_ranger_addr)){
        this->SetError(-1);
        return;
    }


    // Find out which position3d I'll subscribe to
    if (cf->ReadDeviceAddr(&(this->position3d_addr), section,
                                                    "requires", PLAYER_POSITION3D_CODE, -1, NULL) != 0)
```

```
        {
            this->SetError(-1);
            return;
        }

        // Find out which ranger I'll subscribe to
        if (cf->ReadDeviceAddr(&(this->ranger_addr), section,
                                                                "requires", PLAYER_RANGER_CODE, -1, NULL) != 0)
        {
            this->SetError(-1);
            return;
        }

        //prepare data
        memset(&ranger_data_range, 0, sizeof(ranger_data_range));
        ranger_data_range.ranges_count = 4;
        ranger_data_range.ranges = new double[ranger_data_range.ranges_count];
        for(unsigned int i = 0; i < ranger_data_range.ranges_count; i++){
            ranger_data_range.ranges[i] = -1;
        }
        memset(&ranger_data_geom, 0, sizeof(ranger_data_geom));
        memset(&position3d_data_state, 0, sizeof(position3d_data_state));

        puts("LionHell_robot instance Constructed");
}

LionHell_robot:: ~LionHell_robot(void)
{
        std::cout<<std::endl<<"LionHell_robot instance Destructed"<<std::endl;
}

int LionHell_robot::MainSetup(){
        puts("LionHell_robot driver initialising");

        // Subscribe to the position3d device
        if(!(this->position3d_dev = deviceTable->GetDevice(this->ranger_addr)))
        {
            PLAYER_ERROR("unable to locate suitable position3d device");
            return(-1);
        }
        if(this->position3d_dev->Subscribe(this->InQueue) != 0)
        {
            PLAYER_ERROR("unable to subscribe to position3d device");
            return(-1);
        }

        // Subscribe to the ranger device
        if(!(this->ranger_dev = deviceTable->GetDevice(this->ranger_addr)))
        {
            PLAYER_ERROR("unable to locate suitable ranger device");
            return(-1);
        }
        if(this->ranger_dev->Subscribe(this->InQueue) != 0)
        {
            PLAYER_ERROR("unable to subscribe to ranger device");
            return(-1);
        }

        // Here you do whatever else is necessary to setup the device, like open and
        // configure a serial port.

        puts("LionHell_robot driver ready");
        return(0);
}

void LionHell_robot::MainQuit(){
        puts("Shutting LionHell_robot driver down");

        // Unsubscribe from the position3d
        this->position3d_dev->Unsubscribe(this->InQueue);

        // Unsubscribe from the ranger
        this->ranger_dev->Unsubscribe(this->InQueue);

        // Here you would shut the device down by, for example, closing a
        // serial port.
}

void LionHell_robot::Main(void){
        while(1){
```

```
          pthread_testcancel();
          std::cout<<std::endl<<"Main"<<std::endl;
          ProcessMessages();
          PutData();
          usleep(10000);
       }
}

void LionHell_robot::PutData ( void ){
       std::cout<<std::endl<<"PutData"<<std::endl;
       this->Publish(this->my_ranger_addr,
                                        PLAYER_MSGTYPE_DATA, PLAYER_RANGER_DATA_RANGE,
                                        (void*)&ranger_data_range, sizeof(ranger_data_range), NULL);
       this->Publish(this->my_ranger_addr,
                                        PLAYER_MSGTYPE_DATA, PLAYER_RANGER_DATA_GEOM,
                                        (void*)&ranger_data_geom, sizeof(ranger_data_geom), NULL);
       this->Publish(this->my_position3d_addr,
                                PLAYER_MSGTYPE_DATA, PLAYER_POSITION3D_DATA_STATE,
                                        (void*)&position3d_data_state, sizeof(position3d_data_state), NULL);
       puts("Publish done");
}


int LionHell_robot::ProcessMessage(QueuePointer & resp_queue,
                                player_msghdr * hdr,
                                void * data)
{
  // Handle new RANGE data from subscribed RANGER interface
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_DATA, PLAYER_RANGER_DATA_RANGE,
                           this->ranger_addr)){
       update_ranger_range(data);
       return(0);
  }
    // Handle new GEOM data from subscribed RANGER interface
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_DATA, PLAYER_RANGER_DATA_GEOM,
                           this->ranger_addr)){
       update_ranger_geom(data);
    return(0);
  }
    // Handle new STATE data from subscribed POSITION3D interface
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_DATA, PLAYER_POSITION3D_DATA_STATE,
                           this->ranger_addr)){
       this->update_position3d_state(data);
    return(0);
  }
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_REQ, PLAYER_RANGER_REQ_GET_GEOM,
                           this->ranger_addr)){
       puts("RANGER GEOM REQ RECVD");
    return(0);
  }
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_REQ, PLAYER_RANGER_REQ_GET_CONFIG,
                           this->ranger_addr)){
       puts("RANGER GET_CONFIG REQ RECVD");
    return(0);
  }
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_REQ, PLAYER_RANGER_REQ_SET_CONFIG,
                           this->ranger_addr)){
       puts("RANGER SET_CONFIG REQ RECVD");
    return(0);
  }
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_REQ, PLAYER_RANGER_REQ_POWER,
                           this->ranger_addr)){
       puts("RANGER POWER REQ RECVD");
    return(0);
  }
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_REQ, PLAYER_RANGER_REQ_INTNS,
                           this->ranger_addr)){
       puts("RANGER INTNS REQ RECVD");
    return(0);
  }
  // Tell the caller that you don't know how to handle this message
  return(-1);
}

void LionHell_robot::update_position3d_state(void* data){
       this->position3d_data_state.pos.px =  ((player_position3d_data_t*)data)->pos.px;
       this->position3d_data_state.pos.py =  ((player_position3d_data_t*)data)->pos.py;
       this->position3d_data_state.pos.pz =  ((player_position3d_data_t*)data)->pos.pz;
       this->position3d_data_state.pos.proll =  ((player_position3d_data_t*)data)->pos.proll;
       this->position3d_data_state.pos.ppitch =  ((player_position3d_data_t*)data)->pos.ppitch;
```

```
        this->position3d_data_state.pos.pyaw =  ((player_position3d_data_t*)data)->pos.pyaw;
        this->position3d_data_state.vel.px =  ((player_position3d_data_t*)data)->vel.px;
        this->position3d_data_state.vel.py =  ((player_position3d_data_t*)data)->vel.py;
        this->position3d_data_state.vel.pz =  ((player_position3d_data_t*)data)->vel.pz;
        this->position3d_data_state.vel.proll =  ((player_position3d_data_t*)data)->vel.proll;
        this->position3d_data_state.vel.ppitch =  ((player_position3d_data_t*)data)->vel.ppitch;
        this->position3d_data_state.vel.pyaw =  ((player_position3d_data_t*)data)->vel.pyaw;
        this->position3d_data_state.stall =  ((player_position3d_data_t*)data)->stall;
};
void LionHell_robot::update_ranger_range(void* data){
        this->ranger_data_range.ranges_count = ((player_ranger_data_range_t*)data)->ranges_count;
        for(unsigned int i = 0 ; i < this->ranger_data_range.ranges_count; i++ ){
           this->ranger_data_range.ranges[i] = ((player_ranger_data_range_t*)data)->ranges[i];
        }
}
void LionHell_robot::update_ranger_geom(void* data){
        this->ranger_data_geom.pose.px =  ((player_ranger_geom_t*)data)->pose.px;
        this->ranger_data_geom.pose.py =  ((player_ranger_geom_t*)data)->pose.py;
        this->ranger_data_geom.pose.pz =  ((player_ranger_geom_t*)data)->pose.pz;
        this->ranger_data_geom.pose.proll =  ((player_ranger_geom_t*)data)->pose.proll;
        this->ranger_data_geom.pose.ppitch =  ((player_ranger_geom_t*)data)->pose.ppitch;
        this->ranger_data_geom.pose.pyaw =  ((player_ranger_geom_t*)data)->pose.pyaw;
        this->ranger_data_geom.size.sw =  ((player_ranger_geom_t*)data)->size.sw;
        this->ranger_data_geom.size.sl =  ((player_ranger_geom_t*)data)->size.sl;
        this->ranger_data_geom.size.sh =  ((player_ranger_geom_t*)data)->size.sh;
};

Driver* LionHell_robot_Init(ConfigFile* cf, int section){
        std::cout<<std::endl<<"LionHell_robot - Init"<<std::endl;
        return (Driver*) (new LionHell_robot(cf, section));
}

void LionHell_robot_Register(DriverTable* table)
{
 table->AddDriver("LionHell_robot", LionHell_robot_Init ) ;
}

extern "C" {
        int player_driver_init(DriverTable* table){
           puts("doing player_driver_init");
           PLAYER_MSG0 (1, "Registering LionHell_robot driver.");
           LionHell_robot_Register(table);
           return 0 ;
        }
}
```

Configuration Files

This files are used as parameters of ***player*** command, that execute the Player server.

Example:

$ > *player LionHell_robot.cfg*

Configuration file for ***LionHell_pos.so*** plugin
Filename: ***LionHell_pos.conf***

```
driver
(
  name "LionHell_pos"
  provides ["position3d:0"]
  plugin "libplayerLionHell_pos"
)
```

C++ Code
Text 5: Position Plugin - Configuration File Example

Configuration file for ***LionHell_ranger.so*** plugin
Filename: ***LionHell_ranger.conf***

```
driver
(
  name "LionHell_ranger"
  provides ["ranger:0"]
  plugin "libplayerLionHell_ranger"
)
```

C++ Code

Text 6: Ranger Plugin - Configuration File Example

Configuration file for ***LionHell_robot.so*** plugin
Filename: ***`LionHell_robot.conf`***

```
driver
(
  name "LionHell_ranger"
  provides ["ranger:0"]
  plugin "libplayerLionHell_ranger"
)

driver
(
  name "LionHell_pos"
  provides ["position3d:0"]
  plugin "libplayerLionHell_pos"
)

driver
(
        name "LionHell_robot"
        plugin "libplayerLionHell_robot"
        provides ["position3d:1" "ranger:1"]
        #provides ["position3d:1"]
        requires ["position3d:0" "ranger:0"]
        #requires ["position3d:0"]
)
```

C++ Code

Text 7: Robot Plugin - Configuration File Example