Fakultät Informationstechnik

Master in Informationstechnik

Autonome Mobile Roboter - AMR

Final Project - Walking Robot

1220717	Miguel A. Moya Renteria.	mamr23@gmail.com
1210442	Christian R. Jiménez G.	chjimenez@me.com
1210443	Nicolás Faundes	faundez.nicolas@gmail.com

Prof. Dr. Thomas Ihme

Mannheim - February 9, 2014
WS 2013/14

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last thirty years the state of robotic technology has increased its development with applications mainly in industries such as automobile, agriculture, military among others. In the future, Walking robots could offer assistance to a spectrum of applications such as handicapped persons, monotonous tasks, and many other areas of application.

Figure 1.1: PENTA robot by US-ARMY

Figure 1.2: Onda Robot Asimo

" *Compared to wheeled robots, legged robot are more capable of travelling on uneven terrain. Therefore, the design, development and control of legged robots are critical topics in robotics field and a lot of research have been carried on these topics. Till now, hundreds of legged robots have been created over recently 4 decade.*

Compared to other legged robot, biped robots are more dexterous and have higher mobility especially in complicated environments with obstacles. For this reason, applying the biped in the human working environment is promising and, developing biped robots and associated control algorithms are meaningful."[1]

[1] Biped Robots: A brief Review
`http://quickwiki.com/biped_robots.pdf`

1.1 Goals

1. Calibrate the biped robot motors environment.
2. Make a new cable for the RS-485 Interface.
3. Set the Dynamixel-Driver for ROS.
4. Create a Control-Node that works in ROS-Hydro for the PC Computers and ROS-Groovy for the Raspberry environment.
5. Build a C++ that controls the motors through the publication of commands to topics.
6. Build a friendly environment for new students, that allows them to test various motion algorithms.
7. Simulate a motion algorithm in Matlab.
8. Implement a motion algorithm in ROS environment [1].
9. Write a friendly Report, that works as a tutorial.

1.2 Motivation

The main motivation of the project is to create a basic but complete implementation of a motion algorithm. On this basis the following groups of students can provide feedback to the robot through various sensors, such as pressure sensors, cameras, ultrasound sensors, etc, using techniques of visual recognition, position in space, and solutions to avoid obstacles.

Chapter 2

Hardware: Dynamixel Motors

One of the most important parts of a robotic project is the hardware. It is the first part to consider in the design because it sets several features of the project. How fast the robot can react, which maximal speed it can reach or which physical limitations it has, are some good examples of characteristics which the hardware usually settles. Looking from another angle, the physical components actually establish limits in the develop of a project, that is the main reason why hardware has to be choose since the begging of the project.

2.1 The Dynamixel RX-28 Series

The robot of the project (codename "Armstrong") is conformed by 21 motors from ROBOTIS company, but just 20 of them are connected to the communication bus. All the motors are from the same series, the Dynamixel RX-28 (figure2.1). The communication Bus in Armstrong uses the standard RS485 to interconnect all the motors. In order to communicate with the settled bus a cable with a USB-RS285 converter was used.

Figure 2.1: Motor Dynamixel RX-28

A pretty strong aspect in the dynamixel series is, that every motor provides a lot of information about the state of it, data like temperature, present position, present speed and voltage are always accessible. It is also possible to set parameters like ID, Baud rate, speed and range of movement by accessing the Bus (via Instruction Packet [2]). The table 2.1 list and describe the most useful features for the project, the rest of them can be found in the *Dynamixel RX-28 User's Manual* or in the e-Manual of the RX-28 in the ROBOTIS website.

The right configuration of the features mentioned in the table 2.1 is vital for the project. The ID of each motor must be unique, if two motors with the same ID are connected, a packet collision will occur and network problem will take place. The factory default setting ID is 1 for all the motors. A similar case is the Baud rate setting, if different Baud Rates are set in the motors and the Bus, the communication with the motor/Bus will be impossible. It is important to consider that the factory default for the Baud Rate is 57642 bps.

Name	Description	Access
ID	Shows the number with which the motor is identified, it can be set by the user. The configurable value is between 1 and 255.	Read and write
Baud rate	Establish the communication speed with/of the motor. It can be set to values between 7343bps to 1 Mbps. The Baud rate value is setup by changing the value of the register 4 in the EEPROM in each motor, and it can be estimated with the following formula [2]:Speed(BPS) = 2000000/(Data+1)	Read and write
Operation Mode	Shows the mode in which the motor is working. There are just tow possibilities: wheel-type or joint type. This parameter can be changed by modifying the registers CW & CCW angle limit (Registers 6 to 9) iin the EEPROM	Read and write
Speed	It is the moving speed to th goal position. The range and the unit of the value may vary depending of the operation mode [2].It can be set by modifying the values in the Registers 32 and 33 in the RAM of each motor.	Read and write.

Table 2.1: Main features of the Dynamixel RX-28 series used set in the project.

Once the ID and the Baud rate are set in a way that all the motors are communicated with the Bus, the next important configuration is the operation mode of the motor. This setup takes a very important place in the configuration because it affects the range of movement of the robot. When using the wheel-type mode, the motor will be able to move free 360°but the speed control is not available. If the joint mode is used, a zone of 60°will be unaccessible for the motor. This consideration is very important while assembling the robot, **it may cause blocking of necessary movements!!**. The figure 2.2 shows the range of motion when the joint type mode is used.

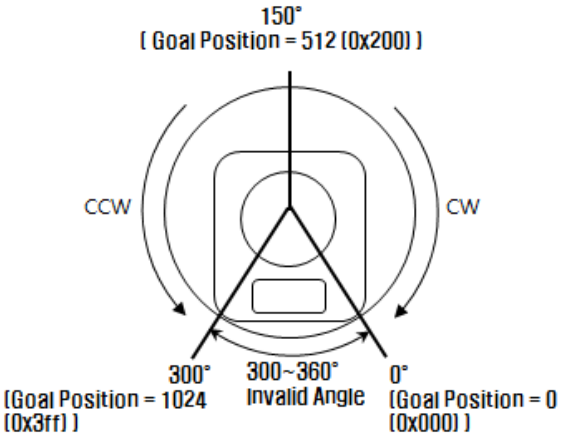

Figure 2.2: Range of motion of a Dynamixel RX-28 in the joint type operation mode.(front side)

The last parameter to consider is the speed. This parameter can be set by changing the values of the registers 32 and 33 in the RAM of the motor (which means it can be changed on-line). To know the set speed of the motor, the value of the registers must be multiplied by 0.111 rpm, if a valule set is 0, the motor will work at the maximal possible speed. The maximal speed of the RX-28 without load is 85rpm when Vin=18.5v (maximal voltage). Important is to consider that the speed is directly proportional to the voltage.

2.2 Motor parameterization in the robot

The previous section mentioned which parameters were important at the time of the setup of the robot. This section shows which configuration was set for each of the named parameters and how was it made.

ID setup

For the ID setup the Dynamixel Wizard tool from the software Roboplus was used. This program runs only in the windows platform, for the project the version 1.0.33 was used. The figure 2.3 shows the start window of the mentioned software. Roboplus offers a full support for the Dynamixel series, with it, the user can red and set all the available parameters in a motor.

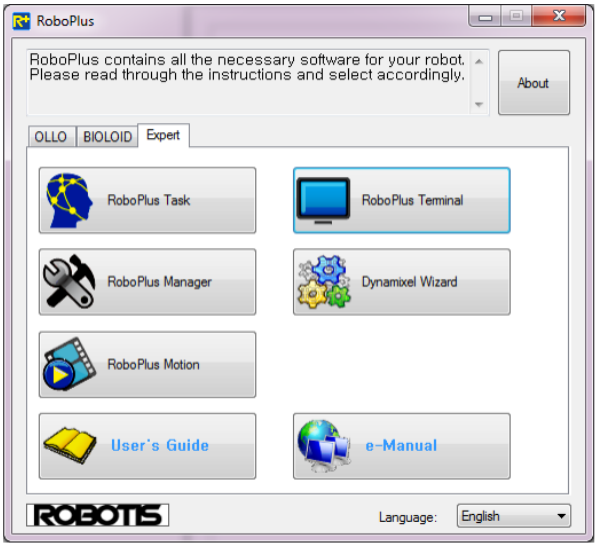

Figure 2.3: Start window of Roboplus

Once the Dynamixel Wizard is opened, a port mus be selected and then open it. The figure 2.4 shows the start window from the Dynamixel Wizard. Once the port is open, the Baud rate of the Bus must be selected and then start the search of the connected motors to the bus (figure 2.5). If it is the first time the motors are read, a Baud rate of 57142 bps must be selected, if the Baud rate setup is unknown, all the possible Baud rates can be selected in the search.

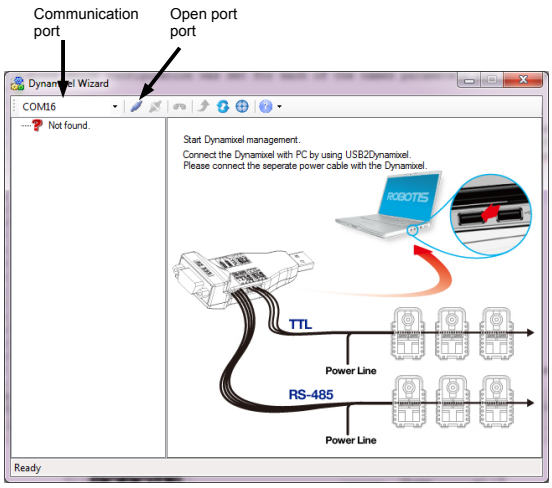

Figure 2.4: Dynamixel Wizard start window.

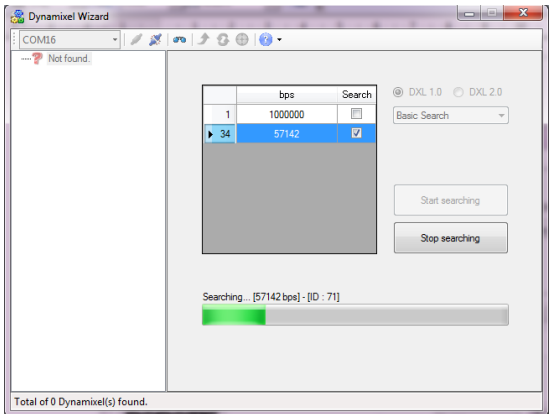

Figure 2.5: Dynamixel Wizard window with one serial Port open

If any motors were found, they will be shown in the left side of the window with its respective ID. In order to see and set the available parameters, the motor must be selected. The figure 2.6 illustrates some of the available configurations of the RX-28 as well as how the ID can be changed.

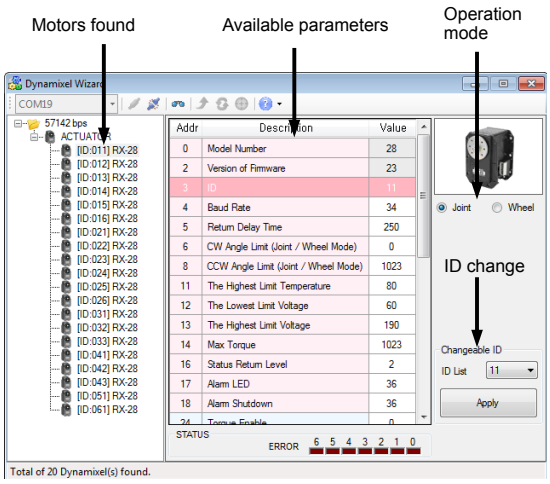

Figure 2.6: Motor setup with the Dynamixel Wizard

The figure2.6 shows too all the motors connected to the bus in Armstrong . The IDs were set in ascending way, each part of the body got a particular number series. For example, the left leg was set with IDs between 11 and 16, while the right got IDs between 21 and 26. The picture 2.7 illustrates the final ID setup of Armstrong.

Baud rate and speed setup

The Baud rate of all the motors remains in the factory default configuration: 57142 bps. The speed wasn't set with Roboplus, it was set directly in ROS, this aspect will be explained in the next chapter.

Operation mode and Position

Because an articulation movement is desired, the operation mode of the motors was set as joint type. As mentioned in the section 2.1, once this mode is set, it is very important to know were the "prohibited zone" in each motor is found and in which direction the motor moves by adding/subtracting grads. The table 2.2 shows the initial position of each motor (when the robot is stood) and in which direction the motor moves when adding/subtracting grads (taking the initial position as reference).

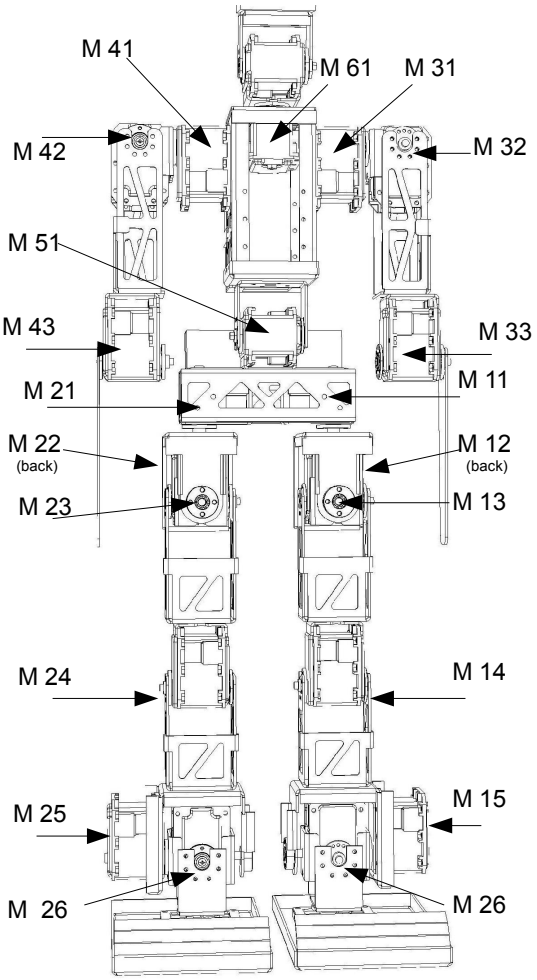

Figure 2.7: Final ID setup of Armstrong

At the beginning of the project, some motors were set in a wrong way. They had their "prohibited zone" in an zone were the motor must move. Due this situation, the position of such motors were re-set, so each motor can archive all the necessary movements in the application. In this reconfiguration, the motors in the arms and head were also considered. The final setup looks for a configuration on which the motors can start without causing any trouble to the robot. Saying that, the initial position to move the robot was set when Armstrong remains stood like the figure 2.7 shows.

Motor ID	**Initial Position**	**Rotation direction when increasing grads**	**Rotation direction when decreasing grads**
11	512	inwards	out
12	444	backwards	forwards
13	512	out	inwards
14	512	forwards	backwards
15	512	up	down
16	338	inwards	out
21	512	out	inwards
22	512	backwards	forwards
23	408	inwards	out
24	512	backwards	forwards
25	512	down	up
26	505	out	Inwards
31	512	Backwards	Forwards
32	512	Down	Up
33	512	Down	Up
41	512	Forwards	Backwards
42	444	Up	Down
43	512	Up	Down
51	512	Down	Up
61	512	Left	Right
62	512	Up	Down

Table 2.2: Initial position and direction of the motors in the robot.

Chapter 3

The Dynamixel Driver in ROS

In the previous chapter it was mentioned how to read and set the different parameters in the motors as well as the initial setup of the complete robot. This chapter will describe the way on which the Dynamixel controller for ROS works and its structure. The driver is contained in a ROS stack called Dynamixel_motor, it does exist another one called *arbotix*, but the Dynaxiel_motor has a better documentation and offers a speed control of the motors (arbotix doesn't [3]). The chapter begins with the installation of the driver in ROS and assumes that the user has a system with an operative ROS Groovy or Hydro installation

3.1 Installation of the Dynamixel motor stack

The installation of the stack which contains the driver in ROS for the Dynamixels series is pretty simple. The packages of the stack are already located in the official ROS repository for both versions: Groovy and hydro. The following command should be run to install the driver in ROS [4]:

```
$ sudo apt-get install ros-distro-dynamixel-motor
```

The first step after the installation is to assure that a communication between linux and the USB-RS485 cable does exist. After the cable has been plugged into the PC, in a terminal the following line should be run:

```
$ ls /dev/ttyUSB*
```

If everything is ok, the output should look like this:

```
$ /dev/ttyUSB0
```

It is also possible that the name of the listed device is different e.g./dev/ttyUSB1, the name of it will depends on how many USB devices are connected to the PC. If the received answer doesn't look like that and the cable works ok in windows, the connection must be reviewed until a similar answer is gotten.

Once the cable has been detected in linux, all the necessary permissions (read and write) must be provided to it. Such permissions can be given through the command:

```
$ sudo chmod 777 /dev/ttyUSB0
```

It is important to remember that, every time the USB RS-485 cable is plugged into the PC, the user must give the permissions to the port. This is because Ubuntu not allows to set permissions fixedly to the group owner of the ports ttyUSBX.

3.2 Structure and test of the Dynamixel Controller package

After the correct installation of the stack , it is possible to use the driver for the dynamixel motors. The installation of the *dynamixel_motor* stack includes a tutorial with which the user can test and know the driver. This tutorial will be used to explain the structure of the driver and how it can be used. The tutorial is located in the *dynamixel_tutorials* folder inside the *dynamixel_motor* stack. To find the folder the user should give in a terminal:

```
$ roscd dynamixel_tutorials
```

Inside of this folder exist all the necessary files to configure every motor and to run the driver which communicates/controls them.

3.2.1 Configuration of the motors

The configuration of the parameters of the servos is made in one file that is part of the *dynamixel_tutorial* package, the `config/dynamixel_joint_controller.yaml`. Inside of it the user can set between other parameters, the name for the controller, the initial-max-min position and the work speed of each motor. Here is how the configuration of a motor in the tutorial looks:

```
1  head_pan_joint:                    //Name of the controller
2      controller :
3          package: dynamixel_controllers
4          module:  joint_position_controller
5          type:  JointPositionController      //Operation mode: Joint type
6      joint_name: head_pan_joint     //Name of the joint
7          joint_speed : 2.0          //Speed of the motor in rad/s
8          motor: id: 1               //Motor controlled by this  controller
9          init : 512                 // Initial  position  of the motor
10         min: 0                     //Minimal reach position of the motor
11         max: 1024                  //Maximal reach position of the motor
```

Listing 3.1: Configuration of a motor in the .yaml file

By modifying this file, is it possible to change the parameters of the motor as well as the configuration of the controller which controls every motor. This setup is very similar for each motor in the project, with the previously mentioned restriction of the IDs, name of the controller and the name of the joint. For the initial position, the user must consider the desirable position of every motor when the application starts. The following listing shows the setup of two motors in the project. The complete configuration for the robot can be found in the Annex A.

```
1
2  controller_m13:
3      controller :
4          package: dynamixel_controllers
5          module:  joint_position_controller
6          type:  JointPositionController
7      joint_name: pan_joint_13
8      joint_speed : 1.0
9      motor:
10         id: 13
11         init : 512
12         min: 0
13         max: 1023
14
15 controller_m14:
16     controller :
17         package: dynamixel_controllers
18         module:  joint_position_controller
19         type:  JointPositionController
20     joint_name: pan_joint_14
21     joint_speed : 1.0
22     motor:
23         id: 14
```

```
24          init : 512
25          min: 0
26          max: 1023
```

Listing 3.2: Configuration of two motors for the project

3.2.2 Initialization of driver and controller

Once the configuration of the motors has been made, it is necessary to run the driver itself and the controllers for every motor in Armstrong. It is made by running two *.launch* files located in the launch folder in the *dynamixel_tutuorials* package. In this folder, two files are found: *controller_manager.launch* and *controller_spawner.launch*. The controller_manager is the one that runs the driver itself and establish the communication with the bus of the robot. The listing 3.3 shows the content of this file.

```
1  <!-- -*- mode: XML -*- -->
2
3  <launch>
4      <node name="dynamixel_manager" pkg="dynamixel_controllers" type="controller_manager.py" required="true"
           output="screen">
5          <rosparam>
6              namespace: dxl_manager
7              serial_ports :
8                  pan_tilt_port :
9                      port_name: "/dev/ttyUSB0"
10                     baud_rate: 57142.0
11                     min_motor_id: 1
12                     max_motor_id: 65
13                     update_rate: 20
14         </rosparam>
15     </node>
16 </launch>
```

Listing 3.3: Controller_manager.launch

In this configuration theres are two vital aspects:

1. The name of the port for the connection must match with the USB RS485 cable. (this case `/dev/ttyUSB0`)

2. The Baud rate setting must match with the motors/bus Baud Rate (this case 56142.0 bps). It is also important that the value is a float number.

The range of IDs (min/max_motor_id) specify the range on which the driver will look for motors, if this range is smaller as the IDs of the motors, the motors will not be detected by the driver.

Once the file has been changed with the adequate values, the file must be launched to initialize the driver:

```
$ roslaunch dynamixel_tutorials controller_manager.launch
```

If everything is ok, the user must get an output similar to the figure 3.1. As can be appreciated, by launching this file, two nodes are started, the **rosout** (the core service) and the **dynamixel_manager**.

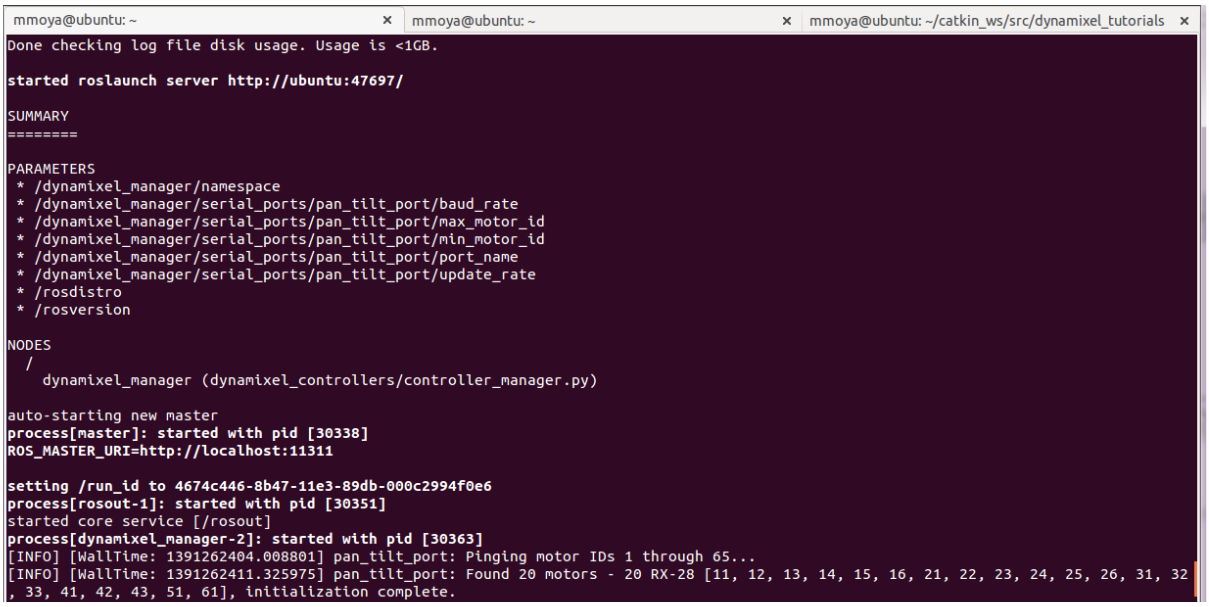

Figure 3.1: Output when running the controller_manager.launch

By looking at the bottom part of the fig. 3.1, it is possible to see how many motors the driver found in the Bus, which model are them and the ID they have. In case of an error, a red text will be displayed in this window. This process **must** be always open during the control of the motors, because of that, a new terminal must be opened to launch or run any other files. The user must not kill this process!!

Another important process which takes place when launching the *controller_manger* file, is that it creates a topic that allows to read the state of the motors. It is possible to identify this topic by running `ostopic list` in a terminal. Then the output should be as follows:

```
/motor_states/pan_tilt_port
/rosout
/rosout_agg
```

The only topic that some connection with the motors hat is `/motor_states/pan_tilt_port`. In order to read the actual state of the motors, run the next command:

```
$ rostopic echo /motor_states/pan_tilt_port
```

By doing that, all the available information (name, Motor ID, temperature, goal position, current position,error, etc.) of each motor will be shown on the screen and will be updated in real time. The fig.3.2 shows the output of the command mentioned before.

To get a better idea of how this "monitoring" on the motors works, it also possible to run the rqtgraph tool. The fig. 3.3 shows its output in this case.

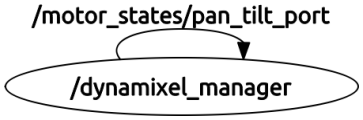

Figure 3.3: Rqt output of the topic */motor_states/pan_tilt_port* and the node *dynamixel_manager*

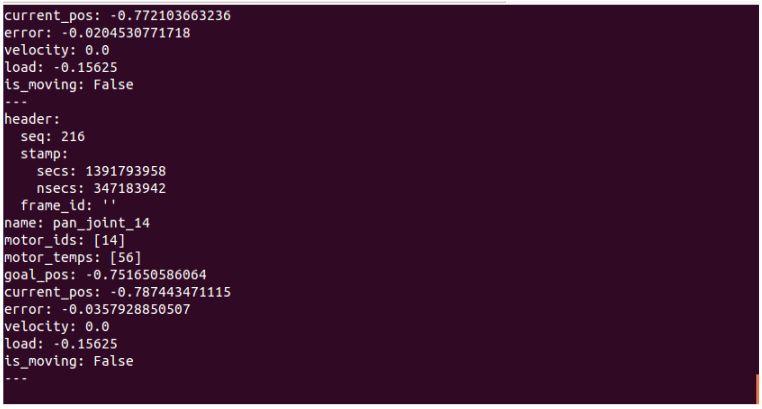

Figure 3.2: Echo from the topic */motor_states/pan_tilt_port*

Once the driver runs, it is possible to get information of every motor, but still impossible to control them. In order to archive gain control of the motors, a controller for the motors should be used. This is done by launching the other file in the *dynamixel_tutorials/launch* folder, the *controller_spawner.launch* file which is listed in the listing 3.4.

```
1  <!-- -*- mode: XML -*- -->
2
3  <launch>
4      <!-- Load controller configuration to parameter server -->
5      <rosparam file="$(find dynamixel_tutorials)/config/dynamixel_joint_controllers.yaml" command="load"/>
6
7      <!-- start specified joint controllers -->
8      <node name="dynamixel_controller_spawner" pkg="dynamixel_controllers" type="controller_spawner.py"
9            args="--manager=dxl_manager
10                 --port=pan_tilt_port
11                 --type=simple
12       controller_m11
13       controller_m12
14       controller_m13
15       controller_m14
16       controller_m15
17       controller_m16
18       controller_m21
19       controller_m22
20       controller_m23
21       controller_m24
22       controller_m25
23       controller_m26
24       controller_m31
25       controller_m32
26       controller_m33
27       controller_m41
28       controller_m42
29       controller_m43
30       controller_m51
31       controller_m61
32       controller_m62"
33           output="screen"/>
34 </launch>
```

Listing 3.4: Controller_spawner.launch

When launching this file, a controller for every listed motor will be created and launched, making it possible to control the motors by using some topics created by the launch file. In order to add a new controller, it is necessary to include the name of the controller listed in the .yaml file discussed before. If both names does not match, the controller will not be created/started.

To launch this file the user must run the following command in a terminal:

```
$ roslaunch dynamixel_tutorials controller_spawner.launch
```

After the `roslaunch` command is successfully executed. The screen will show which controllers were spawn. The figure 3.4 shows a successful initialization of several motors and the failed process for the motor with ID 62.

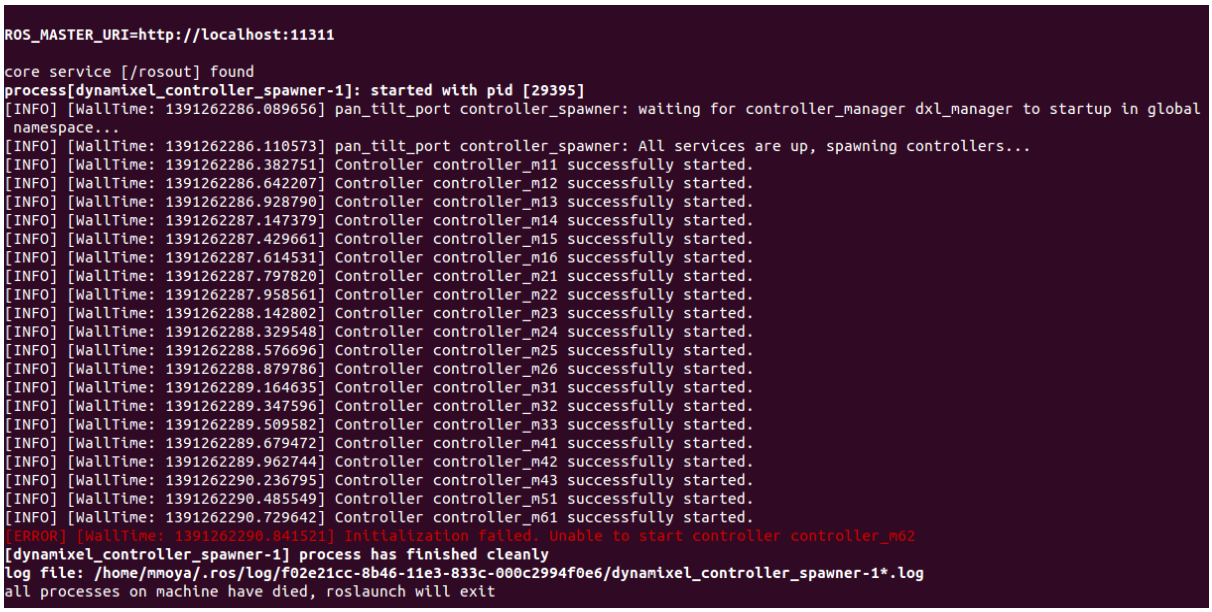

Figure 3.4: Output when running the controller_manager.launch

After launching this file two topics are created for each motor. One to command the movement and other to read the status of the motor (id, goal, position, error, speed, load, voltage, temperature, and a moving status), the difference between this topic and the one created by *controller_manager.launch* is that this new topic can just access one motor, and the other one read at the same time the states of all the motors connected into the bus.

With the `$ rostopic list` command it is possible to see the new topics, the output should show something like:

```
/diagnostics
/motor_states/pan_tilt_port
/rosout
/rosout_agg
/controller_m11/command
/controller_m11/state
/controller_m12/command
/controller_m12/state
          ⋮
/controller_m61/command
/controller_m61/state
```

It is also possible to run the rqtgraph tool to observe how a controller works. The figure 3.5 illustrates how the controller of one motor interacts with the dynamixel_controller node created before and the node of the application (*node_motor* in this case). The creation of the application node will be explained in the following chapter.

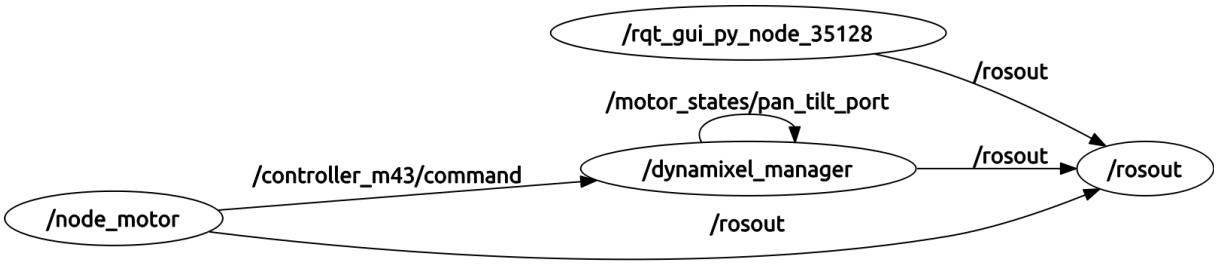

Figure 3.5: Graphical illustration of the interaction of the controller of the motor 14.

To move a particular motor to a new position, the topic `/controller_mxx/command` must be used and to know the temperature or the position of a particular motor, the topic `/controller_mxx/state` should be used.

To give a position to a motor it is necessary to know the variable type that the topic use, for this, the command `$ rostopic type /controller_m11/command` should be run.

The output will show:
`std_msgs/Float64`

Then, a 64 floating number message must be published to move the motor to an specific position, the given value must be in radians. It is possible to move one motor through by using the ROS commands to publish message into a topic:

```
$ rostopic pub /controller_mxx/command std_msgs/Float64 -- -0.785
```

Where -0.785 represents $-45°$ of motion for the motor mxx, starting from the initial position (listed in the .yaml file) as reference for the rotation.

The use of the `rostopic pub` through the command line is impractical to establish initial conditions or movement patterns, so it is necessary to create an application (package-node) that allows the robot to be controlled through a program, this will be described in the next chapter.

3.3 The dynamixel driver in a Raspberry Pi and ROS

In the previous sections it was explained how to install, configure and use the dynamixel stack (driver) en ROS when using a computer with Ubuntu installed. One of the goals of this project was to make it possible to run the software of the application in a Raspberry Pi. Due to the Raspberry Pi design, it is impossible to install any distribution of Ubuntu on it, but it is possible to install a similar linux distribution on it: Raspbian. Raspbian is a debian based linux distribution (Ubuntu is also a debian based distribution) optimized to work in the Raspberry Pi. It is actually better, there is the possibility to install ROS on it.

The project got a Raspbian distribution with ROS groovy already installed. It is important to mention that not all the ROS groovy packages are available in the ROS-Raspbian repository, a status list can be found in:

`http://64.91.227.57/repos/rospbian/debbuild/groovy.html`.

By looking within this packages comes out that the *dynamixel_motor* stack is not available in the repository for Raspbian, so it must by hand installed. The instructions to get the stack working on the Raspberry PI are the following:

1. Download the stack packages from source code: `https://github.com/arebgun/dynamixel_motor`.

2. Copy all the downloaded files into the src folder of your workspace, usually: `user/home/catkin_ws/src/`.

3. Build each packages using ***rosmake*** function in the following order:
   ```
   $ rosmake dynamixel_driver dynamixel_msgs
   $ rosmake dynamixel_motor dynamixel_controllers
   $ rosmake dynamixel_tutorials
   ```

4. Index `devel/setup.bash` so ROS can find the packages:
   ```
   $ source ./catkin_ws/devel/setup.bash
   ```

5. Test the installation, after running the command `$ roscd dynamixel_tutorials`, the terminal should be in something like: `user/home/catkin_ws/src/dynamixel_tutorials $`.

If all the previous steps were successful the *dynamixel_motor* stack is correctly installed in the Raspberry Pi now. Since this point, the configuration and test of the driver are the same as the mentioned before when computer is used.

Chapter 4

Control node

4.1 Moving the motors using a node

4.1.1 Creating a Node

The creation of the node should be no problem, but for `groovy` or `hydro` environments there is a problem with the new workspace and the dynamixel structure. There is errors when `catkin_make` is used to compile the programs.

Therefore an alternative that works well with the `groovy` environment for the Raspberry or the `hydro` environment for the new Ubuntu versions is described below. This procedure was tested in both environments and works perfectly.

1. `$ cd ~/catkin_ws/src/`
2. `$ roscreate-pkg node_motor dynamixel_controllers`. Here the new node `node_motor` inherit all the dependencies from `dynamixel_controllers`.
3. `$ rosmake node_motor`. Whenever necessary to compile the programs, this command must be used.

4.1.2 New Program and CMakeLists.txt

The idea of creating a control node, is to write a program that subscribe and publish topics to control the motors. Additionally, it has the ability to communicate with other nodes that have special functions or access external data files to transmits the results to the motors through the Dynamixel-Interface.

In the scope of this project the program can:

- Publish to the topic `\controller_mxx\command` to move the motors
- Groups the motors to emulate human joints
- Set startup sequences for the motors
- Can read from an external file a list of sequences

The calculation of the direct and inverse dynamics are delegated to a second phase, however the information generated in Matlab for the motion-algorithm is given in a .csv format, which the program has the ability to read and execute.

Because the package was not compiled with `catkin_make`, the procedure to include the correct line to compile the codes in CMakeLists.txt showed in the figure 4.1 should be:

- `$ cd ~/catkin_ws/src/node_motor/`
- `$ gedit CMakeLists.txt`

- Add at the end of the file the following line:
  `rosbuild_add_executable(motor_movements_ext src/motor_movements_ext)`
- Save the file
- The compilation process will take place each time it is run the command:
  `$ rosmake node_motor`.

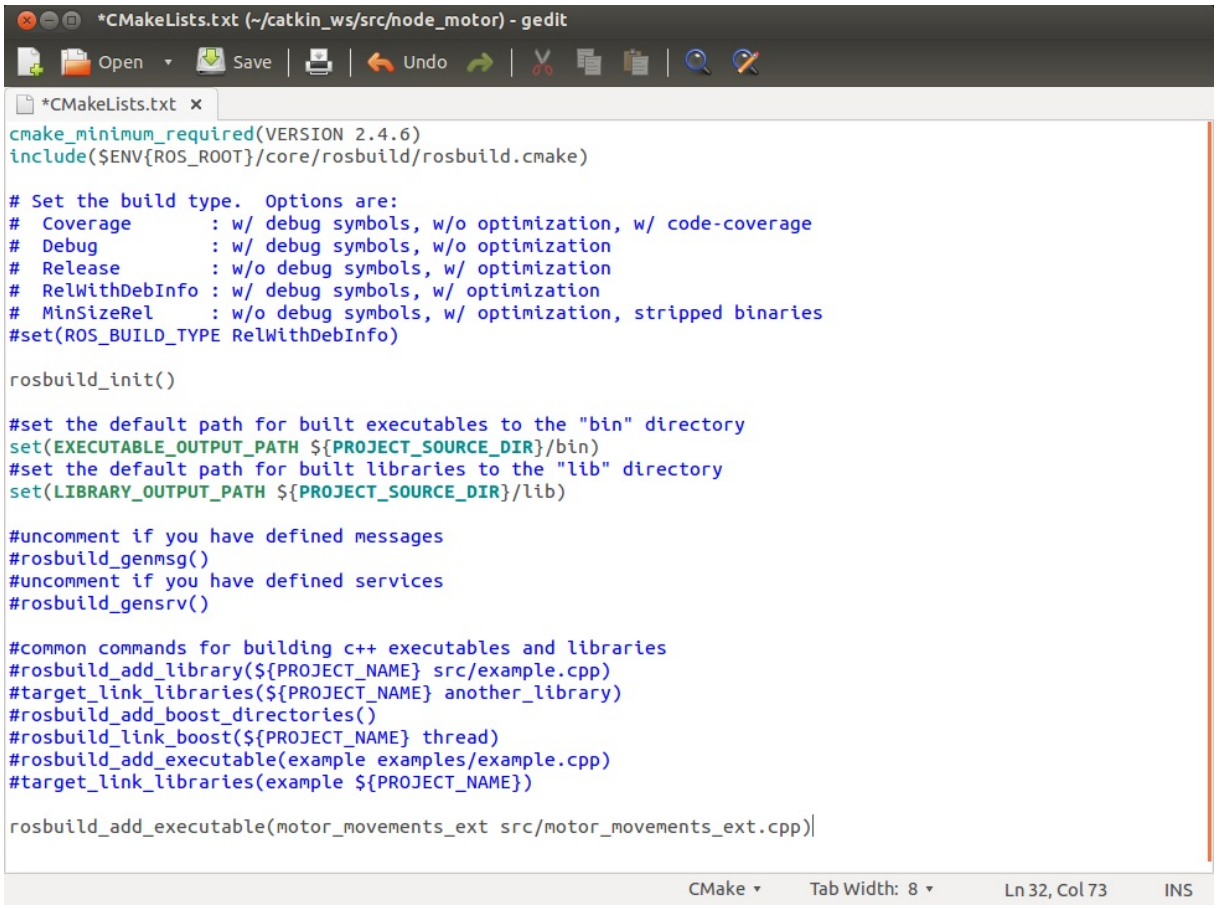

Figure 4.1: CMakeLists.txt

4.1.3 Topology

Figure 4.2 shows the interaction between the node `/motor_node` and each motor topic. The C++ code detailed in this document only has a relationship with the `/controller_mxx/command` through a publish function. However it is possible to access the topic `/controller_mxx/state` through a subscription function.

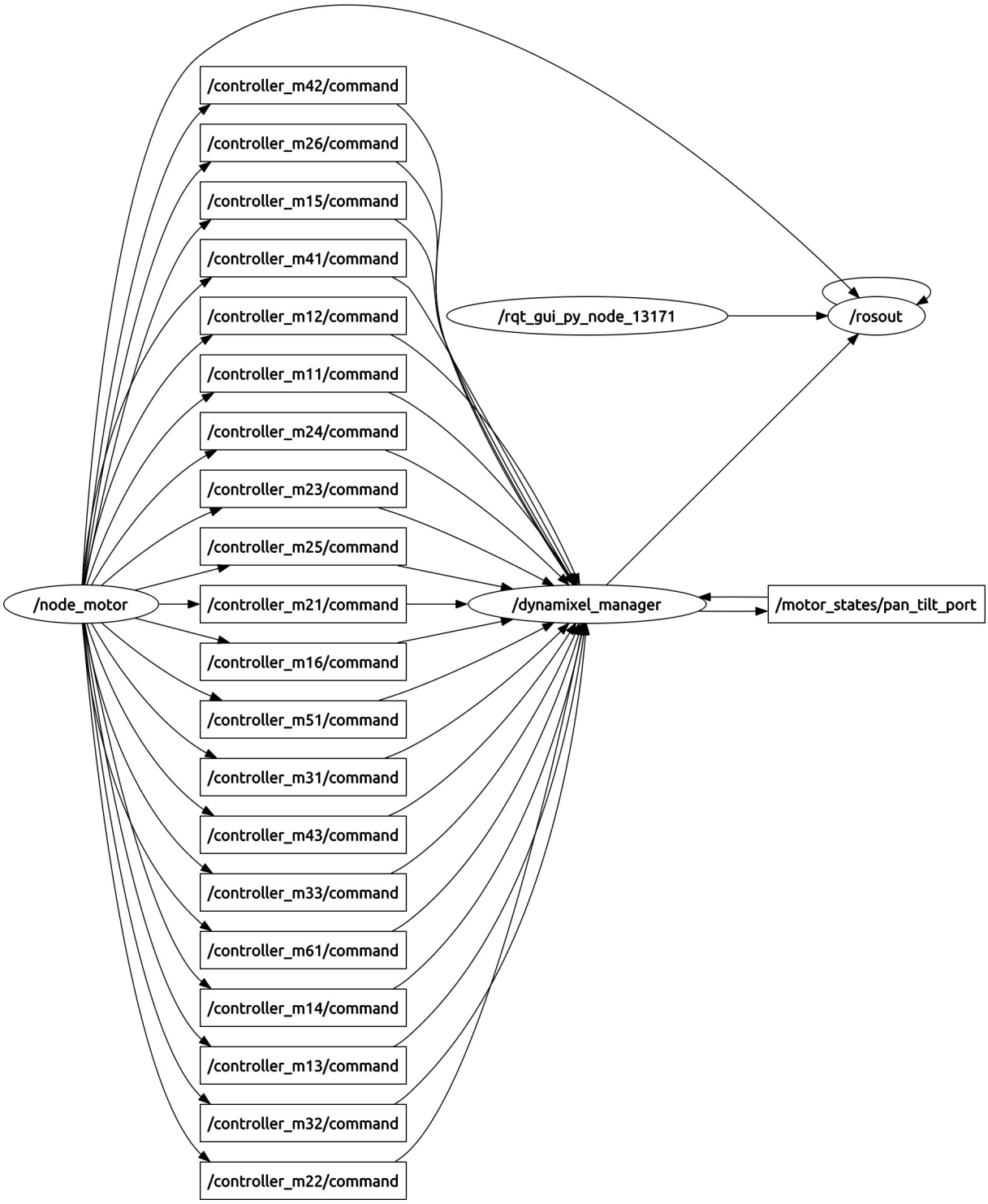

Figure 4.2: Interaction between the Control Node C++ program and the Motor-Topics

4.2 Description of the C++ Code for the Control Node

This section describes how the C++ code controls the motors through the publication of commands to the respective topics. The complete code is referenced in Appendix B. The theory of how a command is published to a topic can be read in the ROS Tutorial named "*Writing a Simple Publisher and Subscriber (C++)*" some of the material of the Tutorial is transcribed and adapted to clarify this code.

4.2.1 How to publish to a topic

```
1 #include<ros/ros.h>
```

ros/ros.h is a convenience include that includes all the headers necessary to use the most common public pieces of the ROS system.

```
2 #include<std_msgs/Float64.h>
```

This includes the std_msgs/Float64 message, which resides in the std_msgs package. This is a header generated automatically from the Float64.msg file in that package.

```
35 class Dynamixel{
36   private:
37     ros::NodeHandle n;
38     ros::Publisher pub_m11;
39     ros::Publisher pub_m12;
40     ...
```

A class Dynamixel will be defined. In a private statement will be declared the NodeHandle and the Publisher required for each motor. In a public statement will be declared all the methods.
NodeHandle is the main access point to communications with the ROS system. The first NodeHandle created will actually do the initialization of the node, and the last one destructed will cleanup any resources the node was using.

```
81 Dynamixel::Dynamixel(){
82   pub_m11 = n.advertise<std_msgs::Float64>("/controller_m11/command",1);
83   pub_m12 = n.advertise<std_msgs::Float64>("/controller_m12/command",1);
84   ...
```

The `advertise()` function is how it tell ROS that it want to publish on a given topic name. This invokes a call to the ROS master node, which keeps a registry of who is publishing and who is subscribing. After this `advertise()` call is made, the master node will notify anyone who is trying to subscribe to the topic name, and they will in turn negotiate a peer-to-peer connection with this node.

`advertise()` returns a Publisher object which allows it to publish messages on that topic through a call to `publish()`. Once all copies of the returned Publisher object are destroyed, the topic will be automatically unadvertised.

The second parameter to `advertise()` is the size of the message queue used for publishing messages. If messages are published more quickly than we can send them, the number here specifies how many messages to buffer up before throwing some away.

The Code described tells the master that it is going to be publishing a message of type `std_msgs/Float64` on the topic `/controller_mxx/command`. This lets the master tell any nodes listening on `/controller_mxx/command` that it is going to publish data on that topic.

`NodeHandle::advertise()` returns a `ros::Publisher` object, which serves two purposes:

1. it contains a `publish()` method that lets to publish messages onto the topic it was created with, and
2. when it goes out of scope, it will automatically unadvertise.

```
125     std_msgs::Float64 aux;
126     aux.data = z*3.14/180;
127     pub_m61.publish(aux);
```

Finally it is possible to publish to a topic declaring a temporary variable of type `std_msgs:: Float64`, and then assign its value to aux.data to finally got published with the `publish()` method.

4.2.2 Joint functions

Now that it is possible to publish via C++ a data to a topic. It is important to create functions that allows to read the code easily. Therefore was chosen a function for each joint. The parameters of the functions represents a motor in a specified coordinate for a joint. All the info is detailed in Figures 4.3, 4.4 and Table 4.1.

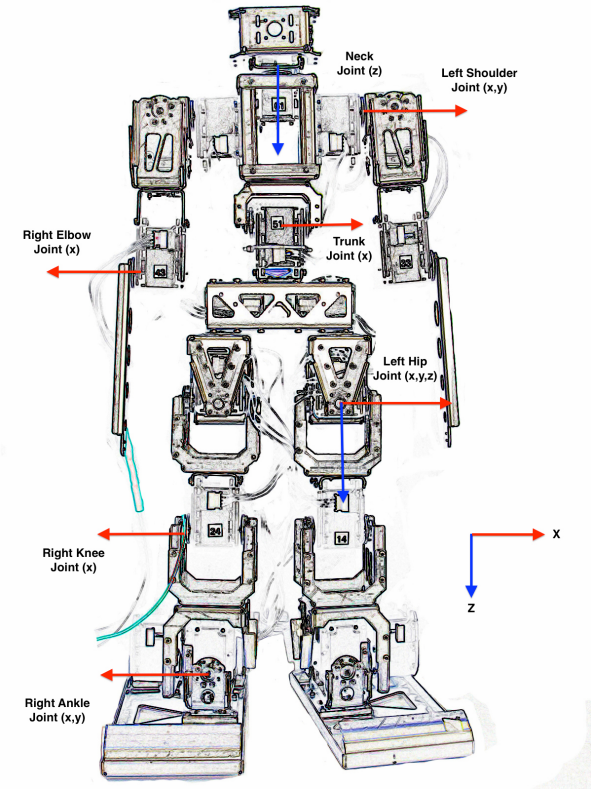

Figure 4.3: Joint-Functions - Front View

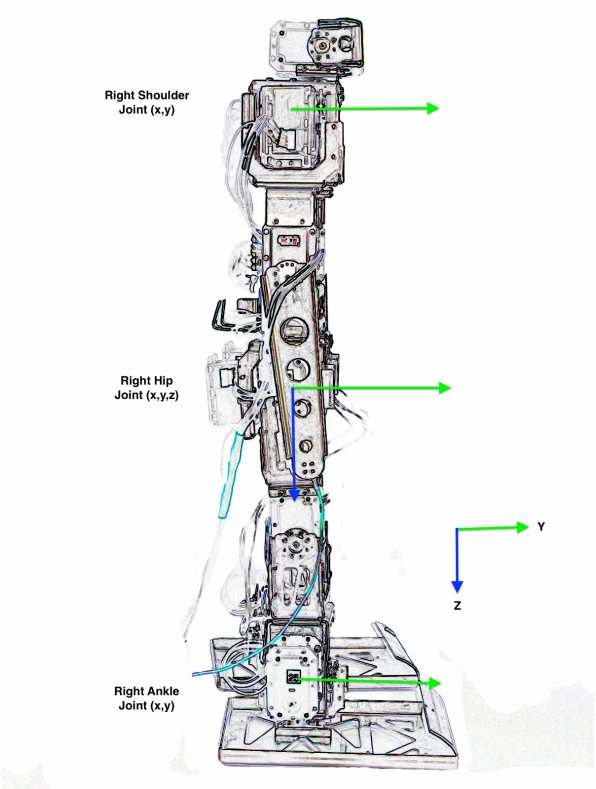

Figure 4.4: Joint-Functions - Lateral View

Joint-Function	X-Coord	Y-Coord	Z-Coord
`moveMotors_neck_joint(z)`			m61
`moveMotors_camera_joint(x)`	m62		
`moveMotors_trunk_joint(x)`	m51		
`moveMotors_left_shoulder_joint(x,y)`	m31	m32	
`moveMotors_left_elbow_joint(x)`	m33		
`moveMotors_right_shoulder_joint(x,y)`	m41	m42	
`moveMotors_right_elbow_joint(x)`	m43		
`moveMotors_left_hip_joint(x,y,z)`	m12	m13	m11
`moveMotors_left_knee_joint(x)`	m14		
`moveMotors_left_ankle_joint(x,y)`	m15	m16	
`moveMotors_right_hip_joint(x,y,z)`	m22	m23	m21
`moveMotors_right_knee_joint(x)`	m24		
`moveMotors_right_ankle_joint(x,y)`	m25	m26	

Table 4.1: Motors by specific function and coordinates

This makes it easy to create sequences of movements, thinking in joints not in motors. Additionally it is a simple way to abstract the robot when it is desired to produce and review pattern sequences.

4.2.3 Initial Position

The Robot was calibrated so that the initial positions of the motors (all the joint parameters in 0) energize the Robot in the position of the Figures 4.3, 4.4.
But it is possible to define different initialization sequences. For the walking algorithm of this report, the following sequence positions was chosen and the resulting robot position is shown in Figures 4.5 and 4.6:

Joint-Function	X-Coord	Y-Coord	Z-Coord
`moveMotors_neck_joint(z)`			0
`moveMotors_camera_joint(x)`	0		
`moveMotors_trunk_joint(x)`	25		
`moveMotors_left_shoulder_joint(x,y)`	0	0	
`moveMotors_left_elbow_joint(x)`	0		
`moveMotors_right_shoulder_joint(x,y)`	0	0	
`moveMotors_right_elbow_joint(x)`	0		
`moveMotors_left_hip_joint(x,y,z)`	-27.666	0.619	0
`moveMotors_left_knee_joint(x)`	-42.992		
`moveMotors_left_ankle_joint(x,y)`	15.326	-0.619	
`moveMotors_right_hip_joint(x,y,z)`	-14.746	5.2503	0
`moveMotors_right_knee_joint(x)`	41.781		
`moveMotors_right_ankle_joint(x,y)`	-27.035	-5.250	

Table 4.2: Angles in Degrees for a particular initial function

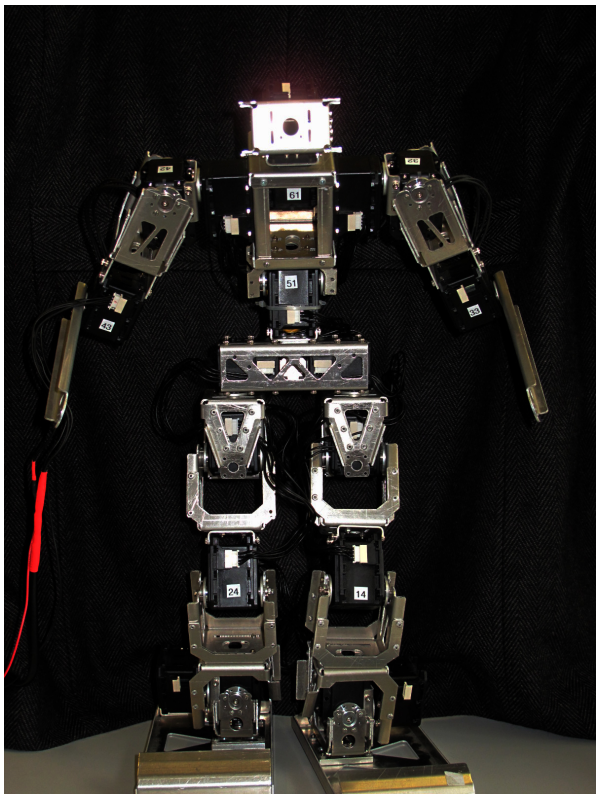

Figure 4.5: Initial-Position - Front View

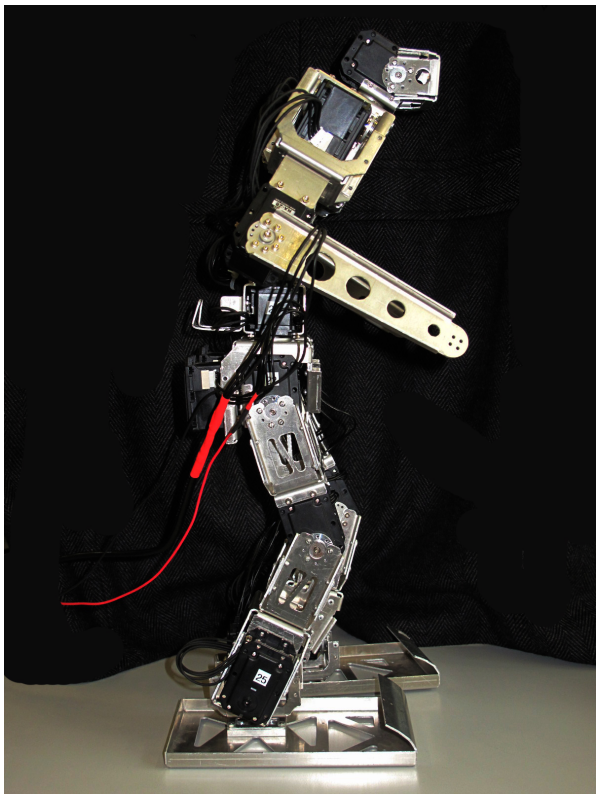

Figure 4.6: Initial-Position - Lateral View

4.2.4 Structure to read and execute an external file

This C++ code allows the node to read external data from a standard format (.csv). This data could be generated by different movement patterns algorithms execute it in programs like Matlab, Mathematica, Maple, etc. This section will detail the method used to read the file and run it as efficiently as possible.

4.2.4.1 File Definition

For simplicity the .csv (comma separated values) format was chosen. The first line contains the description of each column, 21 columns for 21 motors. The Code knows that the first row must be ignored.
The complete file represents a motion sequence, the code asks for steps, namely, how many times a motion sequence is repeated. The table 4.3 shows a sequence extract generated by Matlab in a format required for the C++ Code. All the angles are in degrees.

m11	m12	m13	m14	m15	m16	m21	m22	...	m62
0	-27.666	0.61939	-42.992	15.326	-0.61939	0	-14.746	...	0
0	-26.595	6.7817	-40.935	14.34	-6.7817	0	-12.398	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	-27.476	-2.9396	-42.628	15.152	2.9396	0	-15.152	...	0

Table 4.3: Excerpt from a sequence generated in MatLab

4.2.4.2 Structure Definition

The process of reading a file is a standard in any programming language, and the present Code uses the same principles. However, due to the execution of the code should be as efficient as possible, a data structure was built to load each file-line in memory, and has the distinction of being accessed through a vector class.

In a Raspberry this process may be slow in *loading time*, but *runtime* is as efficient as a normal computer, and this is the goal.

```
10 typedef struct
11 {
12     double m11;
13     double m12;
14     double m13;
15     double m14;
16     double m15;
17     double m16;
18     double m21;
19     double m22;
20     double m23;
21     double m24;
22     double m25;
23     double m26;
24     double m31;
25     double m32;
26     double m33;
27     double m41;
28     double m42;
29     double m43;
30     double m51;
31     double m61;
32     double m62;
33 }structmotoren;
```

From line 10 to 33 in the C++ Code is defined the structure, each structure element represent the data in degrees for each motor properly identified as:
`double mxx;`

```
331 structmotoren winkelstruktur;
```

`structmotoren` is the definition of the structure, `winkelstruktur` will be invoked in the code. Namely, a `winkelstruktur` of type `structmotoren`.

```
334 vector<structmotoren> wv; /* wv = winkelvektoren */
```

`wv` is a vector of structures of type `structmotoren`. This class provides access to each structure in the way that is done with a vector, with the particularity that the size is undefined. For example, To access the data of the motor `m26` in the second structure it is needed only:

```
wv[1].m26
```

Important to notice that the first element of the vector begins with 0. This way of accessing the data through a structure is not only very fast, but it also allows an easy understanding of the code.

4.2.4.3 Reading the File

The process of reading a file is done by default, so only the particular code will be clarify. In the code the syntax `std::` is used but the programmer can really ignore the syntax because it was defined a `namespace` for that purpose:

```
using namespace std;
```

```
104 /* ------------------------------------*/
105 // String to Double Function
106 /* ------------------------------------*/

108 double string_to_double (const std::string& s)
109 {
110     std::istringstream i(s);
111     double x;
112     if (!(i>>x))
113       return 0;
114     return x;
115 }
```

All the data read from the file are of type `string`, to send data to the topic it is required a function that converts the `string` into the type that was defined in the structure. The above function does the job.

```
344 std::ifstream file ("/home/cjimenez/catkin_ws/src/node_motor/bin/winkel_f.csv");
345     std::string line;

347     getline(file, line);
348     /*Here is read the first line of the parameters file.
349     The first line of the file has the ID of the Motors*/
350     while(getline(file, line))
351     {
352         Vector_Length_ ++;

354         std::stringstream iss(line);
355         std::getline(iss,StringValue_,',');
356         winkelstruktur.m11 = string_to_double(StringValue_);

358         std::getline(iss,StringValue_,',');
359         winkelstruktur.m12 = string_to_double(StringValue_);

361         ...
```

The first line is read but ignored, because it contains the labels. In a possible second version of the Code, this line could be useful to identify the motors.

`Vector_Length` count how many steps there are in a sequence.

Finally, each element of the line is read and stored in the data structure through:

```
winkelstruktur.mxx = string_to_double(StringValue);
```

```
415         ...
416         std::getline(iss,StringValue_);
417         winkelstruktur.m62 = string_to_double(StringValue_);

419     /* ---- Here is included the structure in a vector ----- */
420     wv.push_back(winkelstruktur);
421     /*-----------------------------------------------------------*/

423     }
424     file.close();
425     cout<<"Anzahl von Bewegungen per Schritt: "<<Vector_Length_<<endl;
```

When all the elements were recorded in the structure `winkelstruktur`, it is stored in the vector `wv` through:

```
wv.push_back(winkelstruktur);
```

The process is repeated while there are lines in the file. At the end of the file, it is closed and a message to the console of how many moves will be require for a sequence or step.

4.2.4.4 Executing the sequence

```
427    cout<<"Wie viele Schritte? ";
428        cin >> Schritte_;

430                motors.init_Stand_Motors();
431                loop_rate.sleep();

433                motors.init_Stand_Motors();
434                loop_rate.sleep();

436        while(ros::ok() && Counter_ < Schritte_)
437        {
438            int i =0;
439            for (i=1;i<=Vector_Length_;i++) {
440            motors.moveMotors_neck_joint(wv[i-1].m61);
441            motors.moveMotors_left_shoulder_joint(wv[i-1].m31,wv[i-1].m32);
442            motors.moveMotors_right_shoulder_joint(wv[i-1].m41,wv[i-1].m42);
443            motors.moveMotors_right_hip_joint(wv[i-1].m22,wv[i-1].m23,wv[i-1].m21);
444            motors.moveMotors_right_knee_joint(wv[i-1].m24);
445            motors.moveMotors_right_ankle_joint(wv[i-1].m25,wv[i-1].m26);
446            motors.moveMotors_left_hip_joint(wv[i-1].m12,wv[i-1].m13,wv[i-1].m11);
447            motors.moveMotors_left_knee_joint(wv[i-1].m14);
448            motors.moveMotors_left_ankle_joint(wv[i-1].m15,wv[i-1].m16);
449            motors.moveMotors_left_elbow_joint(wv[i-1].m33);
450            motors.moveMotors_right_elbow_joint(wv[i-1].m43);
451            motors.moveMotors_trunk_joint(wv[i-1].m51);
452            loop_rate.sleep();
453            }

455        Counter_ ++;
456        printf("Schrittnummer #: %d \n",Counter_);


459        }
460    }
```

Because the structure is in memory, it is possible to ask the user how many steps he wants the robot to move. The execution is controlled with a `for` loop.

The program concludes by any of these two alternatives:

1. The number of steps typed by the user was reached.
2. ros::ok() will return false if:
    - a SIGINT is received (Ctrl-C)
    - when the user has been kicked off the network by another node with the same name
    - ros::shutdown() has been called by another part of the application.
    - all ros::NodeHandles have been destroyed

Finally a `ros::Rate` object allows the program code to specify a frequency that it would like to loop at. It will keep track of how long it has been since the last call to Rate::sleep(), and sleep for the correct amount of time.
In the Code the loop_rate was set on 4.5Hz or 222 ms through:

```
ros::Rate loop_rate(4.5);
```

and executed through:

```
loop_rate.sleep();
```

That means that each movement is in one sequence is executed in 222ms. Therefore a sequence of 17 movements will be executed in 3.7s.

Chapter 5

Algorithm for Robot Motion

5.1 Biped-Robot Model

In order to move the biped-robot from one point to another, different stages must be archived. The first part is to provide to the robot a trajectory to follow. It could be a simple case such as a straight line or even more difficult such as a curve line. The trajectory of the robot could be referenced to the Mass of Center, however it is better to split the problem assigning the coordinate references to each foot.

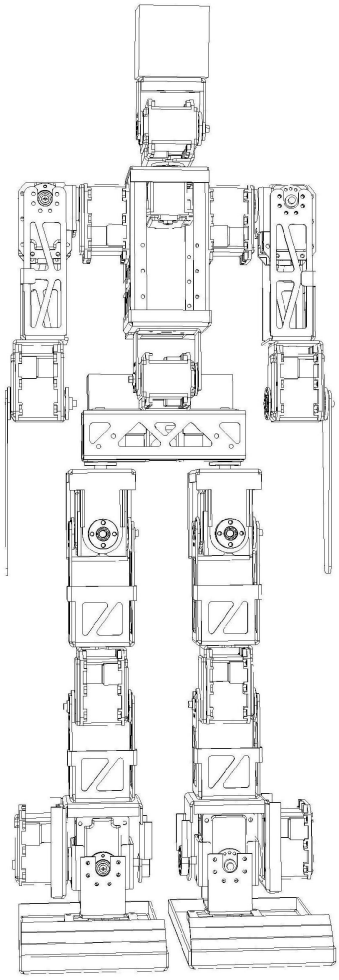

Figure 5.1: Frontal View

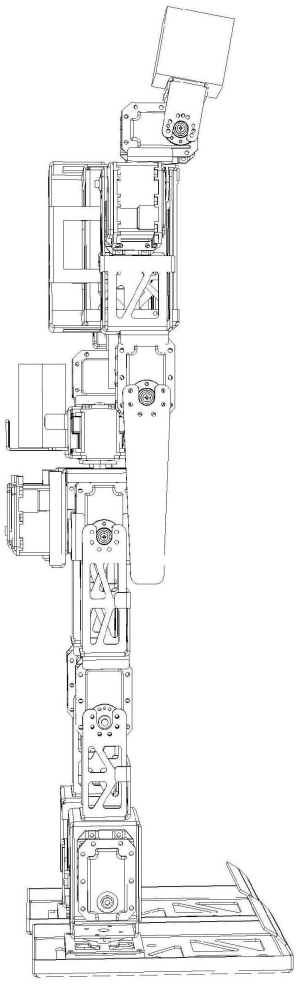

Figure 5.2: Lateral View

The robot is built with 21 servos in different positions and spatial orientation. For the lower part of the body, each leg has 6 servos. The most important angles are q_5, q_4, q_3 which correspond to the servos m_{15},

m_{14}, m_{13} in the left leg, and the motors m_{25},m_{24},m_{23} in the right leg respectively. Those angles are the most important due the relation with the work when making a step in forward direction. The angles q_1, q_2, q_6 are more related to the lateral movements and to the algorithm that holds the equilibrium balance. Those angles correspond to the to the servos m_{11}, m_{12}, m_{16} and to the m_{21}, m_{22}, m_{26} for the left and right legs respectively.

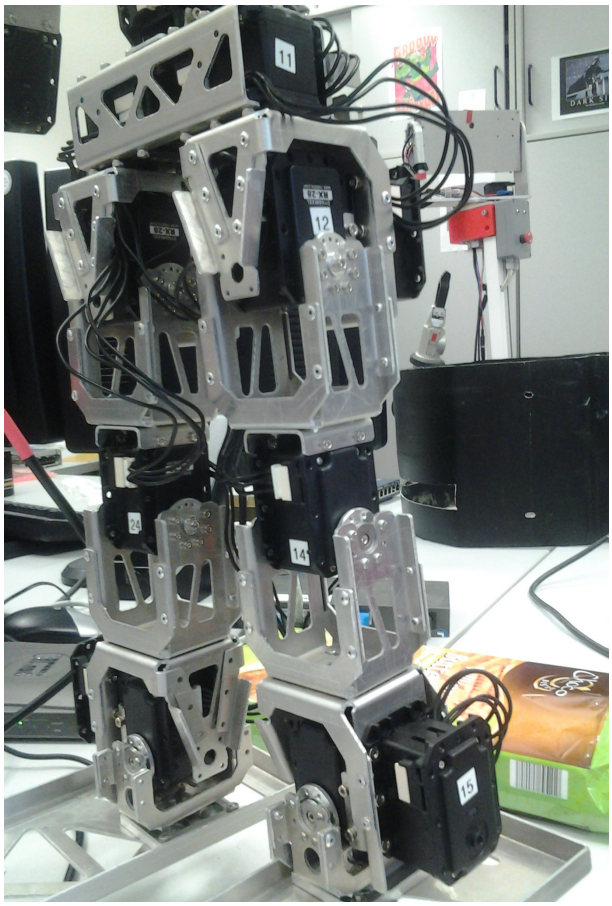

Figure 5.3: Lateral left view of the Robot

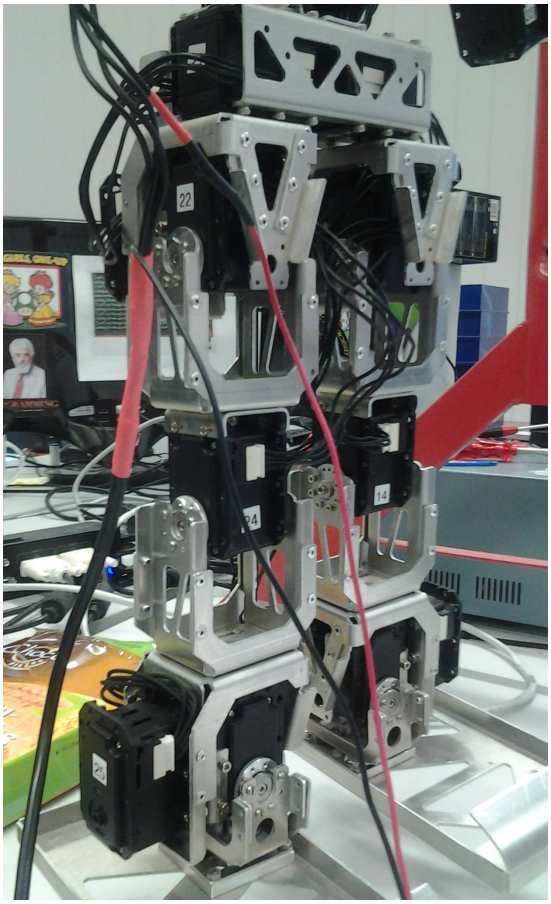

Figure 5.4: Lateral right view of the Robot

To simplify the model, the angles of the other 9 servos (m_{31}, m_{32}, m_{33}, m_{41}, m_{42}, m_{43}, m_{51}, m_{61} and m_{62}) which correspond to the upper body of the robot, are considered to be 0. The main part in the proyect is the walking algorithm model which is addressed to the left and right legs.

5.2 Algorithms

5.2.1 Trajectory Walking

The first task is to implement a suitable algorithm to move the biped-robot from one point to another without lossing the equilibrium of the robot. Analyzing the leg movements of an average person while he walks, it is possible to perceive that the movements occur in different stages. In the figure 5.5 are described the different parameters related with the mechanical model representation of a biped-robot, particularly based in the Robot codename "Armstrong" of the Informatics Department at the Hs-Mannheim.

The parameters used to develop the robot algorithm are defined as follow:

- s_{step} is the lateral distance walked by one step in the y axis.
- l_{step} is the horizontal distance walked by one step in the x axis.

- d_{step} is the fixed height, in which the upper body is lowered in the z axis.
- h_{step} is the highest point that reach the foot in the z axis.
- d_{leg} is the complete sum distance of the links L1, L2, L3 and L4 of the robot.
- t_{step} is the time of required to complete a single step.

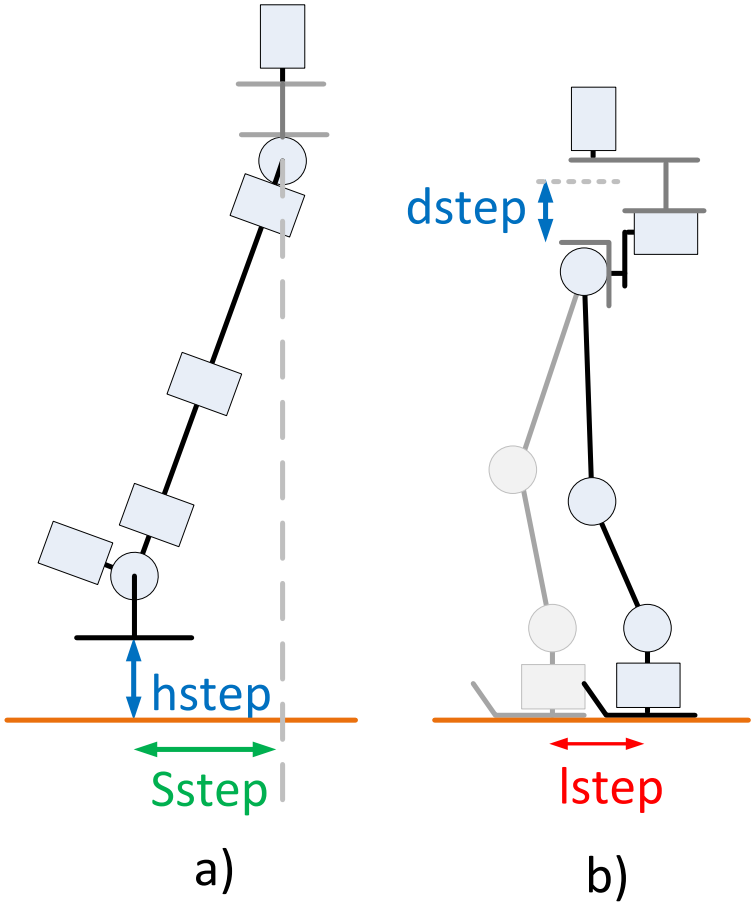

Figure 5.5: Diagram walking biped-robot parameters

The developed algorithm gets the spatial position of each step in Cartesian coordinates. The angles (α, β and γ) addressed to the both foots are defined to 0. For each half cycle the algorithm executes four stages, with the goal of walking one step and keep the equilibrium in range of the center of mass of the robot. The variable p with a value of 4, complete a half cycle and for a value of 8 complete a cycle.

Phase 1 ($withk = 4 \cdot p - 3$)

$$x_{B1,i} = x_{B1,i-1} \qquad x_{B2,i} = x_{B2,i-1}$$

$$y_{L1,i} = y_{L1,i-1} + (-1)^k \cdot s_{step} \qquad y_{L2,i} = y_{L2,i-1} - (-1)^k \cdot s_{step}$$

$$z_{1,i} = z_{L1,i-1} \qquad z_{L2,i} = z_{L2,i-1}$$

Phase 2 ($withk = 4 \cdot p - 2$)

$$x_{L1,i} = x_{L1,i-1} - l_{step}/2 \qquad x_{L1,i} = x_{L2,i-1} + l_{step}/2$$

$$y_{L1,i} = y_{L1,i-1} + \left(\frac{(-1)^k \cdot s_{step}}{(d_{leg} - d_{step})} \cdot h_{step}\right) \qquad y_{L2,i} = y_{L2,i-1} - \left(\frac{(-1)^k \cdot s_{step}}{(d_{leg} - d_{step})} \cdot h_{step}\right)$$

$$z_{L1,i} = z_{L1,i-1} \qquad z_{L2,i} = z_{L2,i-1} + h_{step}$$

Phase 3 $(withk = 4 \cdot p - 1)$

$$x_{L1,i} = x_{L1,i-1} - l_{step}/2 \qquad x_{L2,i} = x_{L2,i-1} + l_{step}/2$$

$$y_{L1,i} = y_{L1,i-1} + \left(\frac{(-1)^k \cdot s_{step}}{(d_{leg} - d_{step})} \cdot h_{step}\right) \qquad y_{L2,i} = y_{L2,i-1} - \left(\frac{(-1)^k \cdot s_{step}}{(d_{leg} - d_{step})} \cdot h_{step}\right)$$

$$z_{L1,i} = z_{L1,i-1} \qquad z_{L2,i} = z_{L2,i-1} - h_{step}$$

Phase 4 (with $k = 4 \cdot p$)
table

$$x_{L1,i} = x_{L1,i-1} \qquad x_{L2,i} = x_{L2,i-1}$$

$$y_{L1,i} = y_{L1,i-1} + (-1)^k \cdot s_{step} \qquad y_{L2,i} = y_{L2,i-1} - (-1)^k \cdot s_{step}$$

$$z_{L1,i} = z_{L1,i-1} \qquad z_{L2,i} = z_{L2,i-1}$$

After of each half cycle (p=4), the algorithm interchanges the variable for both legs. The algorithm for left and right legs interchange the variable in the algorithm, thus procedure is defined by the equations 5.1 and 5.2 for the left and right leg respectively.

$$x_{LL,i} = \begin{cases} x_{L,i}, & \text{if i is even} \\ x_{L,i}, & \text{if i is odd} \end{cases} \tag{5.1}$$

$$x_{RL,i} = \begin{cases} x_{L,i}, & \text{if i is odd} \\ x_{L,i}, & \text{if i is even} \end{cases} \tag{5.2}$$

In order to analyze the performance of the position algorithm, the following parameter were chose.

Parameter	t_{step} **[ms]**	d_{step} **[mm]**	l_{step} **[mm]**	c_{step} **[mm]**	h_{step} **[mm]**	s_{step} **[mm]**
Value	2000	15	40	4	20	20

Table 5.1: Parameter walking algorithm

.

In the figure 5.6 are shown the positions of the left and right foot in spatial Cartesian coordinates.

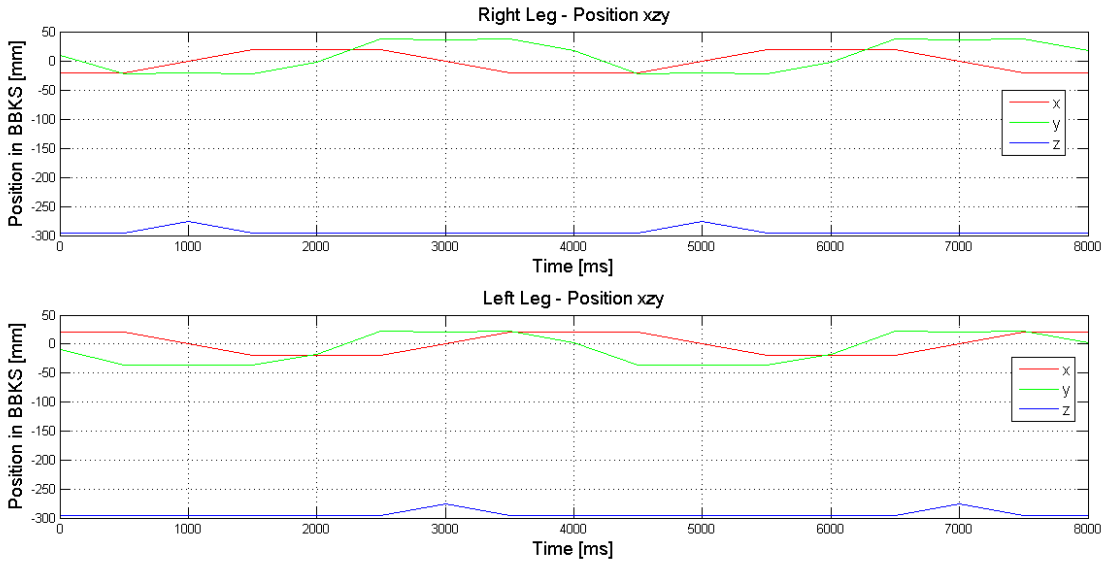

Figure 5.6: Position of left and right legs in Cartesian coordinates

5.2.2 Direct Kinematics

The Direct Kinematics Matrix is a method to transform the angles of a chain of DOF positions in a point into a Cartesian position. This Matrix gives as output the translational matrix which contains the position of the point (p_x,p_y,p_z).

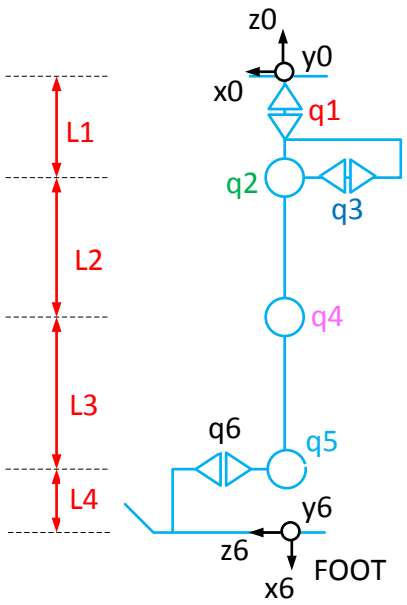

Figure 5.7: DH transformation position_feet

The Denavit-Hartenberg-Parameter matrix of the leg of the robot is given by the table 5.2. The values for the application can be estimated by using the Matlab algorithm *"Foot position"* in the appendixes section.

	θ_i	d_i	a_i	α_i
1	q_1(90°)	-L1	0	+90 °
2	q_2(-90°)	0	0	-90 °
3	q_3(0°)	0	L2	0°
4	q_4(0°)	0	L3	0°
5	q_5(0°)	0	0	+90°
6	q_6(0°)	0	L4	0°

Table 5.2: Denavit-Hartenberg-Parameter for a single leg

The transition matrices which refer to the Denavit-Hartenberg are given by:

$$A_1^0 = \begin{bmatrix} cos(q_1) & 0 & sin(q_1) & 0 \\ sin(q_1) & 0 & -cos(q_1) & 0 \\ 0 & 1 & 0 & -L1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1 = \begin{bmatrix} cos(q_2) & 0 & -sin(q_1) & 0 \\ sin(q_2) & 0 & -cos(q_1) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} cos(q_3) & -sin(q_3) & 0 & 0 \\ sin(q_3) & cos(q_3) & 0 & L_2 sin(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4^3 = \begin{bmatrix} cos(q_4) & -sin(q_4) & 0 & L_3 cos(q_4) \\ sin(q_3) & cos(q_4) & 0 & L_3 sin(q_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5^4 = \begin{bmatrix} cos(q_5) & 0 & sin(q_5) & 0 \\ sin(q_5) & 0 & -cos(q_5) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6^5 = \begin{bmatrix} cos(q_6) & -sin(q_6) & 0 & L_4 cos(q_6) \\ sin(q_6) & -cos(q_6) & 0 & L_4 sin(q_6) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Thus, the translation matrix from the hip to the foot is defined by:

$$T_6^0 = A_1^0 \cdot A_2^1 \cdot A_3^2 \cdot A_4^3 \cdot A_5^4 \cdot A_6^5$$

5.2.3 Inverse Kinematics

The inverse kinematics is given by a matrix which uses the kinematics equations of the robot to determine the joint angular position to provide a desired position of the end-effector (defined such as a hand or a foot). This work is focused in the mechanical model which refers to the legs. The translational matrix representation is defined in general by:

$$T_6^0 = \begin{bmatrix} n_x & O_x & a_x & p_x \\ n_y & O_y & a_y & p_y \\ n_z & O_z & a_z & p_z \\ 0 & 0 & 0 & 1\} \end{bmatrix}$$

The P points (p_x,p_y,p_z) are the Cartesian spatial position of the end-effector (x_6, y_6, z_6) defined for the legs.

5.2.4 DOF Angles of Legs

The performance of the degree of freedom (DOF) angles of each leg is defined by the walking model. It is based on the algorithm and parameters of the end effector positions of the left leg and right legs respectively. This data is generated by using the spatial position of the end effectors joint with the correct translational and inverse matrix. Practical test with the complete algorithm of each foot was applied to the servos of the robot with ROS.

The vector positions of the end effectors are embedded in the translational matrix T_6^0. Those vectors are S_{ll} and S_{ll} for the left and right legs respectively.

$$S_{ll} = \begin{bmatrix} x_{ll} \\ y_{ll} \\ zll \\ \alpha_{ll} \\ \beta_{ll} \\ \gamma_{ll} \end{bmatrix} \qquad S_{rl} = \begin{bmatrix} x_{rl} \\ y_{rl} \\ zrl \\ \alpha_{rl} \\ \beta_r l \\ \gamma_{rl} \end{bmatrix}$$

The trajectory position data of the vectors (S_{ll} and S_{ll}) is considered as an argument by the inverse kinematics transformation matrix, in order to get the information data of angles of each leg. The matrix expression is used as argument by the inverse kinematics, which is embedded in the 4x4 transformation matrix considerate in the end vector spatial position.

With the goal to maintain an stable position in the end effectors and an orientation parallel to the floor. Those angles are set up to zero (α β γ). Also, it makes easier the calculations of the translational matrix.

The angles of the servos of the left and right legs are shown in the figure 5.8.

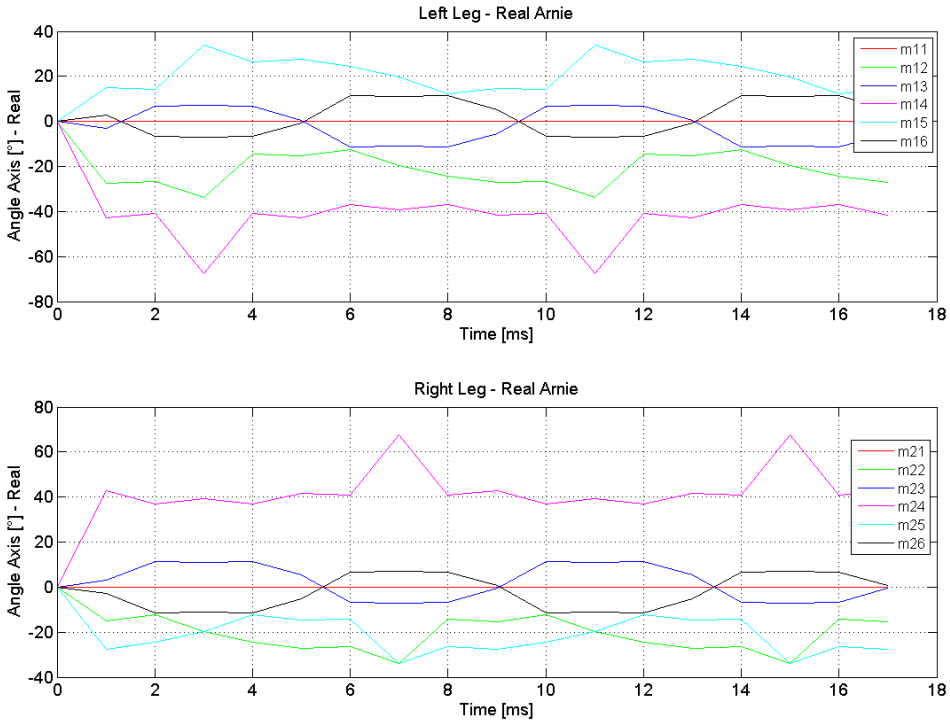

Figure 5.8: Position of leg in Cartesian coordinates

Chapter 6

Next Steps

The present work proposes a model based in an open-loop system, where the concept of feedback is made by theoretically simulations. The basis of the program code is summarized in efficient methods of reading and publication.

But the Dynamixel-Driver together with ROS provide the ability to receive information through the subscription of specific topics. This would transform the system in a closed loop system and will provide the tools to set the appropriate control algorithms. Stability and Recognition Environment, can be achieved through the control system. But before that, the immediate step is to create a C++ code to simplify the subscribe from topics.

Appendix A

Yaml file for the final setup

```
controller_m11:
    controller :
        package: dynamixel_controllers
        module:  joint_position_controller
        type:  JointPositionController
    joint_name: pan_joint_11
    joint_speed : 1.0
    motor:
        id : 11
        init : 512
        min: 0
        max: 1023

controller_m12:
    controller :
        package: dynamixel_controllers
        module:  joint_position_controller
        type:  JointPositionController
    joint_name: pan_joint_12
    joint_speed : 1.0
    motor:
        id : 12
        init : 444
        min: 0
        max: 1023

controller_m13:
    controller :
        package: dynamixel_controllers
        module:  joint_position_controller
        type:  JointPositionController
    joint_name: pan_joint_13
    joint_speed : 1.0
    motor:
        id : 13
        init : 512
        min: 0
        max: 1023

controller_m14:
    controller :
        package: dynamixel_controllers
        module:  joint_position_controller
        type:  JointPositionController
    joint_name: pan_joint_14
    joint_speed : 1.0
    motor:
        id : 14
        init : 512
        min: 0
        max: 1023

controller_m15:
    controller :
```

```
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
        joint_name: pan_joint_15
        joint_speed: 1.0
        motor:
            id: 15
            init: 512
            min: 0
            max: 1023

controller_m16:
        controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
        joint_name: pan_joint_16
        joint_speed: 1.0
        motor:
            id: 16
            init: 338
            min: 0
            max: 1023

controller_m21:
        controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
        joint_name: pan_joint_21
        joint_speed: 1.0
        motor:
            id: 21
            init: 512
            min: 0
            max: 1023

controller_m22:
        controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
        joint_name: pan_joint_22
        joint_speed: 1.0
        motor:
            id: 22
            init: 512
            min: 0
            max: 1023

controller_m23:
        controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
        joint_name: pan_joint_23
        joint_speed: 1.0
        motor:
            id: 23
            init: 408
            min: 0
            max: 1023

controller_m24:
        controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
        joint_name: pan_joint_24
        joint_speed: 1.0
        motor:
            id: 24
            init: 512
```

```
            min: 0
            max: 1023

controller_m25:
      controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
      joint_name: pan_joint_25
      joint_speed: 1.0
      motor:
            id: 25
            init: 512
            min: 0
            max: 1023

controller_m26:
      controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
      joint_name: pan_joint_26
      joint_speed: 1.0
      motor:
            id: 26
            init: 505
            min: 0
            max: 1023

controller_m31:
      controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
      joint_name: pan_joint_33
      joint_speed: 1.0
      motor:
            id: 31
            init: 512
            min: 0
            max: 1023

controller_m32:
      controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
      joint_name: pan_joint_33
      joint_speed: 1.0
      motor:
            id: 32
            init: 512
            min: 0
            max: 1023

controller_m33:
      controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
      joint_name: pan_joint_33
      joint_speed: 1.0
      motor:
            id: 33
            init: 512
            min: 0
            max: 1023

controller_m41:
      controller:
            package: dynamixel_controllers
            module: joint_position_controller
            type: JointPositionController
```

```
    joint_name: pan_joint_33
    joint_speed: 1.0
    motor:
        id: 41
        init: 512
        min: 0
        max: 1023

controller_m42:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: pan_joint_33
    joint_speed: 1.0
    motor:
        id: 42
        init: 444
        min: 0
        max: 1023

controller_m43:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: pan_joint_33
    joint_speed: 1.0
    motor:
        id: 43
        init: 512
        min: 0
        max: 1023

controller_m51:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: pan_joint_33
    joint_speed: 1.0
    motor:
        id: 51
        init: 512
        min: 0
        max: 1023

controller_m61:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: pan_joint_61
    joint_speed: 1.0
    motor:
        id: 61
        init: 512
        min: 0
        max: 1023

controller_m62:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: pan_joint_62
    joint_speed: 1.0
    motor:
        id: 62
        init: 512
        min: 0
        max: 1023
```

Appendix B

C++ Code

```
#include<ros/ros.h>
#include<std_msgs/Float64.h>
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<sstream>
#include<fstream>

using namespace std;
typedef struct
{
    double m11;
    double m12;
    double m13;
    double m14;
    double m15;
    double m16;
    double m21;
    double m22;
    double m23;
    double m24;
    double m25;
    double m26;
    double m31;
    double m32;
    double m33;
    double m41;
    double m42;
    double m43;
    double m51;
    double m61;
    double m62;
}structmotoren;

class Dynamixel{
    private:
        ros::NodeHandle n;
        ros::Publisher pub_m11;
        ros::Publisher pub_m12;
        ros::Publisher pub_m13;
        ros::Publisher pub_m14;
        ros::Publisher pub_m15;
        ros::Publisher pub_m16;
        ros::Publisher pub_m21;
        ros::Publisher pub_m22;
        ros::Publisher pub_m23;
        ros::Publisher pub_m24;
        ros::Publisher pub_m25;
        ros::Publisher pub_m26;
        ros::Publisher pub_m31;
        ros::Publisher pub_m32;
        ros::Publisher pub_m33;
        ros::Publisher pub_m41;
        ros::Publisher pub_m42;
```

```
55          ros::Publisher pub_m43;
56          ros::Publisher pub_m51;
57          ros::Publisher pub_m61;
58          ros::Publisher pub_m62;

60      public:
61          Dynamixel();
62          double string_to_double (const std::string& s);
63          int init_Stand_Motors();
64          int moveMotors_neck_joint(double z);
65          int moveMotors_left_shoulder_joint(double x, double y);
66          int moveMotors_left_elbow_joint(double x);
67          int moveMotors_right_shoulder_joint(double x, double y);
68          int moveMotors_right_elbow_joint(double x);
69          int moveMotors_right_hip_joint(double x, double y, double z);
70          int moveMotors_right_knee_joint(double x);
71          int moveMotors_right_ankle_joint(double x, double y);
72          int moveMotors_left_hip_joint(double x, double y, double z);
73          int moveMotors_left_knee_joint(double x);
74          int moveMotors_left_ankle_joint(double x, double y);
75          int moveMotors_camera_joint(double z);
76          int moveMotors_trunk_joint(double x);

78  };


81  Dynamixel::Dynamixel(){
82      pub_m11 = n.advertise<std_msgs::Float64>("/controller_m11/command",1);
83      pub_m12 = n.advertise<std_msgs::Float64>("/controller_m12/command",1);
84      pub_m13 = n.advertise<std_msgs::Float64>("/controller_m13/command",1);
85      pub_m14 = n.advertise<std_msgs::Float64>("/controller_m14/command",1);
86      pub_m15 = n.advertise<std_msgs::Float64>("/controller_m15/command",1);
87      pub_m16 = n.advertise<std_msgs::Float64>("/controller_m16/command",1);
88      pub_m21 = n.advertise<std_msgs::Float64>("/controller_m21/command",1);
89      pub_m22 = n.advertise<std_msgs::Float64>("/controller_m22/command",1);
90      pub_m23 = n.advertise<std_msgs::Float64>("/controller_m23/command",1);
91      pub_m24 = n.advertise<std_msgs::Float64>("/controller_m24/command",1);
92      pub_m25 = n.advertise<std_msgs::Float64>("/controller_m25/command",1);
93      pub_m26 = n.advertise<std_msgs::Float64>("/controller_m26/command",1);
94      pub_m31 = n.advertise<std_msgs::Float64>("/controller_m31/command",1);
95      pub_m32 = n.advertise<std_msgs::Float64>("/controller_m32/command",1);
96      pub_m33 = n.advertise<std_msgs::Float64>("/controller_m33/command",1);
97      pub_m41 = n.advertise<std_msgs::Float64>("/controller_m41/command",1);
98      pub_m42 = n.advertise<std_msgs::Float64>("/controller_m42/command",1);
99      pub_m43 = n.advertise<std_msgs::Float64>("/controller_m43/command",1);
100     pub_m51 = n.advertise<std_msgs::Float64>("/controller_m51/command",1);
101     pub_m61 = n.advertise<std_msgs::Float64>("/controller_m61/command",1);
102     pub_m62 = n.advertise<std_msgs::Float64>("/controller_m62/command",1);
103 }
104 /* ---------------------------------------*/
105 // String to Double Function
106 /* ---------------------------------------*/

108 double string_to_double (const std::string& s)
109 {
110     std::istringstream i(s);
111     double x;
112     if (!(i>>x))
113       return 0;
114     return x;
115 }

117 /* ---------------------------------------*/
118 // Joint Functions for the Head and Trunk
119 /* ---------------------------------------*/
120 // Neck Horizontal Movement
121 /* ---------------------------------------*/

123 int Dynamixel::moveMotors_neck_joint(double z)
124 {
125     std_msgs::Float64 aux;
126     aux.data = z*3.14/180;
127     pub_m61.publish(aux);
```

```
128     return 1;
129 }
130 /*--------------------------------------*/
131 // Camera Head Movement (Up-Down)
132 /*--------------------------------------*/
133 int Dynamixel::moveMotors_camera_joint(double x)
134 {
135     std_msgs::Float64 aux;
136     aux.data = x*3.14/180;
137     pub_m62.publish(aux);
138     return 1;
139 }
140 /*--------------------------------------*/
141 // Trunk
142 /*--------------------------------------*/
143 int Dynamixel::moveMotors_trunk_joint(double x)
144 {
145     std_msgs::Float64 aux;
146     aux.data = x*3.14/180;
147     pub_m51.publish(aux);
148     return 1;
149 }

151 /*--------------------------------------*/
152 // Joint Functions for the Arms
153 /*--------------------------------------*/
154 // Left Arm
155 /*--------------------------------------*/
156 // Left Shoulder Joint
157 int Dynamixel::moveMotors_left_shoulder_joint(double x, double y)
158 {
159     std_msgs::Float64 aux;
160     aux.data = x*3.14/180;
161     pub_m31.publish(aux);
162     aux.data = y*3.14/180;
163     pub_m32.publish(aux);
164     return 1;
165 }

167 // Left Elbow Joint
168 int Dynamixel::moveMotors_left_elbow_joint(double x)
169 {
170     std_msgs::Float64 aux;
171     aux.data = x*3.14/180;
172     pub_m33.publish(aux);
173     return 1;
174 }
175 /*--------------------------------------*/
176 // Right Arm
177 /*--------------------------------------*/
178 // Right Shoulder Joint
179 int Dynamixel::moveMotors_right_shoulder_joint(double x, double y)
180 {
181     std_msgs::Float64 aux;
182     aux.data = x*3.14/180;
183     pub_m41.publish(aux);
184     aux.data = y*3.14/180;
185     pub_m42.publish(aux);
186     return 1;
187 }

189 // Right Elbow Joint
190 int Dynamixel::moveMotors_right_elbow_joint(double x)
191 {
192     std_msgs::Float64 aux;
193     aux.data = x*3.14/180;
194     pub_m43.publish(aux);
195     return 1;
196 }

198 /*--------------------------------------*/
199 // Joint Functions for the Legs
200 /*--------------------------------------*/
```

```
201 // Left Leg
202 /* ----------------------------------------*/
203 // Left Hip Joint
204 int Dynamixel::moveMotors_left_hip_joint(double x, double y, double z)
205 {
206     std_msgs::Float64 aux;
207     aux.data = x*3.14/180;
208     pub_m12.publish(aux);
209     aux.data = y*3.14/180;
210     pub_m13.publish(aux);
211     aux.data = z*3.14/180;
212     pub_m11.publish(aux);
213     return 1;
214 }
215 // Left Knee Joint
216 int Dynamixel::moveMotors_left_knee_joint(double x)
217 {
218     std_msgs::Float64 aux;
219     aux.data = x*3.14/180;
220     pub_m14.publish(aux);
221     return 1;
222 }
223 // Left Ankle Joint
224 int Dynamixel::moveMotors_left_ankle_joint(double x, double y)
225 {
226     std_msgs::Float64 aux;
227     aux.data = x*3.14/180;
228     pub_m15.publish(aux);
229     aux.data = y*3.14/180;
230     pub_m16.publish(aux);
231     return 1;
232 }
233 /* ----------------------------------------*/
234 // Right Leg
235 /* ----------------------------------------*/
236 // Right Hip Joint
237 int Dynamixel::moveMotors_right_hip_joint(double x, double y, double z)
238 {
239     std_msgs::Float64 aux;
240     aux.data = x*3.14/180;
241     pub_m22.publish(aux);
242     aux.data = y*3.14/180;
243     pub_m23.publish(aux);
244     aux.data = z*3.14/180;
245     pub_m21.publish(aux);
246     return 1;
247 }
248 // Right Knee Joint
249 int Dynamixel::moveMotors_right_knee_joint(double x)
250 {
251     std_msgs::Float64 aux;
252     aux.data = x*3.14/180;
253     pub_m24.publish(aux);
254     return 1;
255 }
256 // Right Ankle Joint
257 int Dynamixel::moveMotors_right_ankle_joint(double x, double y)
258 {
259     std_msgs::Float64 aux;
260     aux.data = x*3.14/180;
261     pub_m25.publish(aux);
262     aux.data = y*3.14/180;
263     pub_m26.publish(aux);
264     return 1;
265 }
266 /* ----------------------------------------*/
267 int Dynamixel::init_Stand_Motors()
268 {
269     std_msgs::Float64 aux;
270     aux.data = 0*3.14/180;
271     pub_m11.publish(aux);
272     aux.data = -27.666*3.14/180;
273     pub_m12.publish(aux);
```

```
274     aux.data = 0.619*3.14/180;
275     pub_m13.publish(aux);
276     aux.data = -42.992*3.14/180;
277     pub_m14.publish(aux);
278     aux.data = 15.326*3.14/180;
279     pub_m15.publish(aux);
280     aux.data = -0.619*3.14/180;
281     pub_m16.publish(aux);
282     aux.data = 0*3.14/180;
283     pub_m21.publish(aux);
284     aux.data = -14.746*3.14/180;
285     pub_m22.publish(aux);
286     aux.data = 5.2503*3.14/180;
287     pub_m23.publish(aux);
288     aux.data = 41.781*3.14/180;
289     pub_m24.publish(aux);
290     aux.data = -27.035*3.14/180;
291     pub_m25.publish(aux);
292     aux.data = -5.250*3.14/180;
293     pub_m26.publish(aux);
294     aux.data = 0*3.14/180;
295     pub_m31.publish(aux);
296     aux.data = 0*3.14/180;
297     pub_m32.publish(aux);
298     aux.data = 0*3.14/180;
299     pub_m33.publish(aux);
300     aux.data = 0*3.14/180;
301     pub_m41.publish(aux);
302     aux.data = 0*3.14/180;
303     pub_m42.publish(aux);
304     aux.data = 0*3.14/180;
305     pub_m43.publish(aux);
306     aux.data = 25*3.14/180;
307     pub_m51.publish(aux);
308     aux.data = 0*3.14/180;
309     pub_m61.publish(aux);
310 return 1;
311 }

313 /* ---------------------------------------*/
314 /* Main Program */
315 /* ---------------------------------------*/

317 int main(int argc,char** argv)
318 {
319     ros::init(argc, argv, "node_motor");
320     Dynamixel motors;
321     string StringValue_;
322     int Counter_ = 0;
323     int Vector_Length_ = 0;
324     int Schritte_ = 0;

326     /* ---------------------------------------*/
327     /* Here is read the angles parameters from a .csv file ,
328     then the info is located in a structure and the structrure
329     is located in a Vector. */

331     structmotoren winkelstruktur;
332     /* Here is defined the structures in a vector, each line is
333        a structure and each structure a position in a vector */
334     vector<structmotoren> wv; /* wv = winkelvektoren */
335     /* ---------------------------------------*/
336     ros::Rate loop_rate(4.5);

338             //motors.init_Stand_Motors();
339             //loop_rate.sleep();

341             //motors.init_Stand_Motors();
342             //loop_rate.sleep();

344     std::ifstream file ("/home/cjimenez/catkin_ws/src/node_motor/bin/winkel_f.csv");
345     std::string line;
```

```
347     getline( file , line );
348     /*Here is read the  first  line  of the parameters file .
349     The first  line  of the  file  has the ID of the Motors*/
350     while( getline ( file , line ))
351     {

353         Vector_Length_ ++;

355         std :: stringstream  iss ( line );
356         std :: getline ( iss ,StringValue_,',');
357         winkelstruktur.m11 = string_to_double(StringValue_);

359         std :: getline ( iss ,StringValue_,',');
360         winkelstruktur.m12 = string_to_double(StringValue_);

362         std :: getline ( iss ,StringValue_,',');
363         winkelstruktur.m13 = string_to_double(StringValue_);

365         std :: getline ( iss ,StringValue_,',');
366         winkelstruktur.m14 = string_to_double(StringValue_);

368         std :: getline ( iss ,StringValue_,',');
369         winkelstruktur.m15 = string_to_double(StringValue_);

371         std :: getline ( iss ,StringValue_,',');
372         winkelstruktur.m16 = string_to_double(StringValue_);

374         std :: getline ( iss ,StringValue_,',');
375         winkelstruktur.m21 = string_to_double(StringValue_);

377         std :: getline ( iss ,StringValue_,',');
378         winkelstruktur.m22 = string_to_double(StringValue_);

380         std :: getline ( iss ,StringValue_,',');
381         winkelstruktur.m23 = string_to_double(StringValue_);

383         std :: getline ( iss ,StringValue_,',');
384         winkelstruktur.m24 = string_to_double(StringValue_);

386         std :: getline ( iss ,StringValue_,',');
387         winkelstruktur.m25 = string_to_double(StringValue_);

389         std :: getline ( iss ,StringValue_,',');
390         winkelstruktur.m26 = string_to_double(StringValue_);

392         std :: getline ( iss ,StringValue_,',');
393         winkelstruktur.m31 = string_to_double(StringValue_);

395         std :: getline ( iss ,StringValue_,',');
396         winkelstruktur.m32 = string_to_double(StringValue_);

398         std :: getline ( iss ,StringValue_,',');
399         winkelstruktur.m33 = string_to_double(StringValue_);

401         std :: getline ( iss ,StringValue_,',');
402         winkelstruktur.m41 = string_to_double(StringValue_);

404         std :: getline ( iss ,StringValue_,',');
405         winkelstruktur.m42 = string_to_double(StringValue_);

407         std :: getline ( iss ,StringValue_,',');
408         winkelstruktur.m43 = string_to_double(StringValue_);

410         std :: getline ( iss ,StringValue_,',');
411         winkelstruktur.m51 = string_to_double(StringValue_);

413         std :: getline ( iss ,StringValue_,',');
414         winkelstruktur.m61 = string_to_double(StringValue_);

416         std :: getline ( iss ,StringValue_);
417         winkelstruktur.m62 = string_to_double(StringValue_);

419     /* ---- Here is included the structure in a vector ----- */
```

```
420     wv.push_back(winkelstruktur);
421     /*-------------------------------------------------------------*/

423     }
424      file . close ();
425     cout<<"Anzahl von Bewegungen per Schritt: "<<Vector_Length_<<endl;
426 /* ----------------------------------------*/
427     cout<<"Wie viele Schritte? ";
428     cin >> Schritte_;

430             motors.init_Stand_Motors();
431             loop_rate . sleep ();

433             motors.init_Stand_Motors();
434             loop_rate . sleep ();

436     while(ros :: ok() && Counter_ < Schritte_)
437     {
438         int i =0;
439         for (i=1;i<=Vector_Length_;i++) {
440         motors.moveMotors_neck_joint(wv[i-1].m61);
441         motors.moveMotors_left_shoulder_joint(wv[i-1].m31,wv[i-1].m32);
442         motors.moveMotors_right_shoulder_joint(wv[i-1].m41,wv[i-1].m42);
443         motors.moveMotors_right_hip_joint(wv[i-1].m22,wv[i-1].m23,wv[i-1].m21);
444         motors.moveMotors_right_knee_joint(wv[i-1].m24);
445         motors.moveMotors_right_ankle_joint(wv[i-1].m25,wv[i-1].m26);
446         motors.moveMotors_left_hip_joint(wv[i-1].m12,wv[i-1].m13,wv[i-1].m11);
447         motors.moveMotors_left_knee_joint(wv[i-1].m14);
448         motors.moveMotors_left_ankle_joint(wv[i-1].m15,wv[i-1].m16);
449         motors.moveMotors_left_elbow_joint(wv[i-1].m33);
450         motors.moveMotors_right_elbow_joint(wv[i-1].m43);
451         motors.moveMotors_trunk_joint(wv[i-1].m51);
452         loop_rate . sleep ();
453         }

455     Counter_ ++;
456     printf("Schrittnummer #: %d \n",Counter_);


459     }
460 }
```

Listing B.1: C++ control node code

Appendix C

Main file of the Matlab algorithm

```
clear all
close all

% DH parameter
% d: it is called the offset, which is the distance along z_(i-1) from
% O(i-1) to intersection of x_i and z_(i-1)
%
% a: it is called the length(link), which is the distance along x_i from
% O_i to intersection of x_i and z_(i-1)
%
% alpha: it is called the twist, which is the angel between z_(i-1)
% measured about x_i
%
% theta: it is called the angle, which is the angle x_(i-1) measured about
% z_(i-1)

%% Possition of each foot
foot_position

%% Model Axis angles
axis_angle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function Real Angles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Real axis angles
axis_angle_real

%% Practical axis angles
axis_angle_practical
```

Appendix D

Foot Position file in the Matlab

```
1
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Function Foot Position Trajectory
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  %% ORIENTATION WALK
7  % X variable: move to forward step
8  % Y variable: move to lateral step
9  % Z variable: move to up step
10
11 %% STEP PARAMETERS
12 parameters
13
14 h_step=20;   % 20 [mm] highest step distance from floor % Z-Y variable
15 d_step=15;   % 40 [mm] highest step distance from hip   % Y variable
16 l_step=40;   % 30 [mm] distance trail of each step      % X variable
17 s_step=20;   % 50 [mm] distance hip of each step        % Y variable
18 t_step=2000; % time [ms] time duration step
19 c_step=4;    % counter step
20
21 % Length Leg
22 L1=70; % 90 or 70
23 L2=100; %95
24 L3=100;
25 L4=40;
26
27 d_leg=L1+L2+L3+L4;
28
29 %% Walking Model Parameters
30
31 P=c_step;
32 %
33 x_b1 = l_step/2;
34 x_b2= -l_step/2;
35
36 y_b1= -9.5;
37 y_b2= 9.5;
38 z_b1= -d_leg+d_step;
39 z_b2= -d_leg+d_step;
40
41 X_ll =x_b1;
42 Y_ll= y_b1;
43 Z_ll= -d_leg+d_step;
44
45 X_rl =x_b2;
46 Y_rl= y_b2;
47 Z_rl= -d_leg+d_step;
48
49 % vector time
50 t=0;
51 ts=0;
52 %
53
54 for p=1:P
```

```
55
56 %% Phase 1
57 %
58 k=p*4-3; % minus start
59 %
60 x_b1_1= x_b1;
61 y_b1_1= y_b1+(-1)^k*s_step;
62 z_b1_1= z_b1;
63 %
64 x_b2_1= x_b2;
65 y_b2_1= y_b2-(-1)^k*s_step;
66 z_b2_1= z_b2;
67 %
68
69     % PAR CASE
70     if  mod(p,2) == 0
71     %
72     x_rl_1 =x_b1_1; y_rl_1=y_b2_1+d_step/2; z_rl_1=z_b1_1;
73     x_ll_1 =x_b2_1; y_ll_1=y_b2_1-d_step/2; z_ll_1=z_b2_1;
74
75     %X_rl=[X_rl,x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
76     X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
77     X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
78
79     % ODD CASE
80     else
81     %
82     x_rl_1 =x_b2_1; y_rl_1=y_b1_1+d_step/2; z_rl_1=z_b2_1;
83     x_ll_1 =x_b1_1; y_ll_1=y_b1_1-d_step/2; z_ll_1=z_b1_1;
84     %
85     %X_rl=[X_rl,x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
86     X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
87     X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
88     %
89     end
90 x_b1=x_b1_1; y_b1=y_b1_1; z_b1=z_b1_1;
91 x_b2=x_b2_1; y_b2=y_b2_1; z_b2=z_b2_1;
92
93 ts=ts+t_step/4;
94 t=[t,ts ];
95
96 %% Phase 2
97 %
98 k=p*4-2; %plus
99 %
100 x_b1_1= x_b1-l_step/2;
101 y_b1_1= y_b1+((-1)^k*s_step*h_step/(d_leg-d_step));
102 z_b1_1= z_b1;
103 %
104 x_b2_1= x_b2+l_step/2;
105 y_b2_1= y_b2-((-1)^k*s_step*h_step/(d_leg-d_step));
106 z_b2_1= z_b2+h_step;
107 %
108     if  mod(p,2) == 0
109     %
110     x_rl_1 =x_b1_1; y_rl_1=y_b2_1+d_step/2; z_rl_1=z_b1_1;
111     x_ll_1 =x_b2_1; y_ll_1=y_b2_1-d_step/2; z_ll_1=z_b2_1;
112     X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
113     X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
114     %
115     else
116     %
117     x_rl_1 =x_b2_1; y_rl_1=y_b1_1+d_step/2; z_rl_1=z_b2_1;
118     x_ll_1 =x_b1_1; y_ll_1=y_b1_1-d_step/2; z_ll_1=z_b1_1;
119     %
120     X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
121     X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
122     %
123     end
124 x_b1=x_b1_1; y_b1=y_b1_1; z_b1=z_b1_1;
125 x_b2=x_b2_1; y_b2=y_b2_1; z_b2=z_b2_1;
126
127 ts=ts+t_step/4;
```

```
t=[t,ts];

%% Phase 3
%
k=p*4-1; %minus
%
x_b1_1= x_b1-l_step/2;
y_b1_1= y_b1+((-1)^k*s_step*h_step/(d_leg-d_step));
z_b1_1= z_b1;
%
x_b2_1= x_b2+l_step/2;
y_b2_1= y_b2-((-1)^k*s_step*h_step/(d_leg-d_step));
z_b2_1= z_b2-h_step;
%

 if  mod(p,2) == 0
      %
      x_rl_1 =x_b1_1; y_rl_1=y_b2_1+d_step/2; z_rl_1=z_b1_1;
      x_ll_1 =x_b2_1; y_ll_1=y_b2_1-d_step/2; z_ll_1=z_b2_1;
      %
      X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
      X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
%
      else
      %
      x_rl_1 =x_b2_1; y_rl_1=y_b1_1+d_step/2; z_rl_1=z_b2_1;
      x_ll_1 =x_b1_1; y_ll_1=y_b1_1-d_step/2; z_ll_1=z_b1_1;
      %
      X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
      X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
      %
      end
x_b1=x_b1_1; y_b1=y_b1_1; z_b1=z_b1_1;
x_b2=x_b2_1; y_b2=y_b2_1; z_b2=z_b2_1;

ts=ts+t_step/4;
t=[t,ts];

%% Phase 4
%
k=p*4; % plus
%
x_b1_1= x_b1;
y_b1_1= y_b1+(-1)^k*s_step;
z_b1_1= z_b1;
%
x_b2_1= x_b2;
y_b2_1= y_b2-(-1)^k*s_step;
z_b2_1= z_b2;
%
      if  mod(p,2) == 0
      %
      x_rl_1 =x_b1_1; y_rl_1=y_b2_1+d_step/2; z_rl_1=z_b1_1;
      x_ll_1 =x_b2_1; y_ll_1=y_b2_1-d_step/2; z_ll_1=z_b2_1;
      %
      X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
      X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
%
      else
      %
      x_rl_1 =x_b2_1; y_rl_1=y_b1_1+d_step/2; z_rl_1=z_b2_1;
      x_ll_1 =x_b1_1; y_ll_1=y_b1_1-d_step/2; z_ll_1=z_b1_1;
      %
      X_rl=[X_rl, x_rl_1 ];  Y_rl=[Y_rl, y_rl_1 ];  Z_rl=[Z_rl, z_rl_1 ];
      X_ll=[X_ll, x_ll_1 ];  Y_ll=[Y_ll, y_ll_1 ];  Z_ll=[Z_ll, z_ll_1 ];
      %
      end

x_b1=x_b2_1;
y_b1=y_b1_1;
y_b2=y_b2_1;
z_b1=z_b1_1;
x_b2=x_b1_1;
```

```
z_b2=z_b2_1;

ts=ts+t_step/4;
t=[t,ts];

end
%% S Vector

[a,b]=size(X_ll);
c=zeros(1,b);

%left leg
S_ll=[X_ll;Y_ll;Z_ll;c;c;c];


%right leg
S_rl=[X_rl;Y_rl;Z_rl;c;c;c];

%% SIMULATION

%PLOT
%% Position of right and left foot
figure(5)
% right leg
subplot(211)
plot(t,X_rl,'-r');
hold on
plot(t,Y_rl, '-g');
hold on
plot(t,Z_rl, '-b');
legend('x','y','z','Location','NorthEast')
ylabel('Position in BBKS [mm]','fontsize',14)
xlabel('Time [ms]','fontsize',14)
title('Right Leg - Position xzy','fontsize',14)
grid on
%
% left leg
subplot(212)
plot(t,X_ll,'-r');
hold on
plot(t,Y_ll, '-g');
hold on
plot(t,Z_ll, '-b');
legend('x','y','z','Location','NorthEast')
ylabel('Position in BBKS [mm]','fontsize',14)
xlabel('Time [ms]','fontsize',14)
title('Left Leg - Position xzy','fontsize',14)
grid on
%
figure(6)
plot(t,X_rl,'-r');
hold on
plot(t,X_ll, '-r');
hold on
plot(t,Y_rl, '-g');
% hold on
plot(t,Y_ll, '-g');
hold on
plot(t,Z_rl, '-b');
hold on
plot(t,Z_ll, '-b');
ylabel('Position in BBKS [mm]','fontsize',14)
xlabel('Time [ms]','fontsize',14)
title('Both Legs - Position xzy','fontsize',14)
grid on
```

Appendix E

Real Angles file in the Matlab

```
1
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Function Real Angles
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  %% Values of Legs
7  %
8  % q1
9  % + -> 2.1(left leg) -> rotate outward
10 % + -> 1.1(right leg) -> rotate inward
11 % q2
12 % + -> 2.3(left leg) -> rotate clock
13 % + -> 1.3(right leg) -> rotate clock
14 % q3
15 % + -> 2.2(left leg) -> rotate down
16 % + -> 1.2(right leg) -> rotate down
17 % q4
18 % + -> 2.4(left leg) -> rotate down
19 % + -> 1.4(right leg) -> rotate up
20 % q5
21 % + -> 2.5(left leg) -> rotate down
22 % + -> 1.5(right leg) -> rotate up
23 % q6
24 % + -> 2.6(left leg) -> rotate clock
25 % + -> 1.6(right leg) -> rotate clock
26
27 % Transformation
28 % 1023(max_angle) 0(min angle)
29
30 [L,O]=size(Theta_rl);
31
32 %%
33 % m11,m12,m13,m14,m15,m16,m21,m22,m23,m24,m25,m26
34 %
35 % Left leg
36 mm11=Theta_ll(:,1)-90*ones(L,1); %0          - OK
37 mm12=Theta_ll(:,2)+90*ones(L,1); %-20        - not
38 mm13=Theta_ll(:,3);             %0          - not
39 mm14=-Theta_ll(:,4);            % -42        - OK
40 mm15=-Theta_ll(:,5);            % -
41 mm16=Theta_ll(:,6);             %
42
43 % Right Leg
44 mm21=Theta_rl(:,1)-90*ones(L,1); % 0         - OK
45 mm22=Theta_rl(:,2)+90*ones(L,1); % -15       - not
46 mm23=Theta_rl(:,3);             % 0          - not
47 mm24=Theta_rl(:,4);             % +42.5      - OK
48 mm25=Theta_rl(:,5);             % -
49 mm26=Theta_rl(:,6);             %
50
51 %% Change Angles
52
53 % Left leg
54 m11=mm11; %0          - OK
```

```
m12=-mm15; %-20          - changed 12 by 13
m13=-mm12; %0            - changed 22 by 23
m14=mm14;                % -42           - OK
m15=-mm13;               % -
m16=-mm16;               %

% Right Leg
m21=mm21; % 0            - OK
m22=mm25; % -15          - changed 22 by 23
m23=-mm22;               % 0             - changed 23 by 22
m24=mm24;                % +42.5         - OK
m25=mm23;                % -
m26=-mm26;               %

%% Values of Upper Body
% m31,m32,m33,m41,m42,m43,m51,m61,m62
P_par=[-30,0,0,-30,0,0,0,0,0];
P_odd =[30,0,0,30,0,0,0,0,0];
%
[L,O]=size(Theta_rl);
%
M_winkel    =[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
%
for  l=1:L
    if  mod(l,2) == 0
        M_wink=[m11(l,:),m12(l,:),m13(l,:),m14(l,:),m15(l,:),m16(l,:),m21(l,:),m22(l,:),m23(l,:),m24(l,:),m25(l,:),
            m26(l,:),P_par];
    else
        M_wink=[m11(l,:),m12(l,:),m13(l,:),m14(l,:),m15(l,:),m16(l,:),m21(l,:),m22(l,:),m23(l,:),m24(l,:),m25(l,:),
            m26(l,:),P_odd];
    end
M_winkel=[M_winkel;M_wink];

end

% Save the angles in a csv Format file
csvwrite('csv_winkel_v2_lstep40.dat',M_winkel)

%%
figure(12)
% right leg
subplot(211)
plot(t,m11','-r');
hold on
plot(t,m12', '-g');
hold on
plot(t,m13', '-b');
hold on
plot(t,m14','-m');
hold on
plot(t,m15', '-c');
hold on
plot(t,m16', '-k');
legend('m11','m12','m13','m14','m15','m16','Location','NorthEast')
ylabel('Angle Axis [ ] - Real','fontsize',14)
xlabel('Time [ms]','fontsize',14)
title('Left Leg - Real Arnie','fontsize',14)
grid on

% left leg
subplot(212)
plot(t,m21','-r');
hold on
plot(t,m22', '-g');
hold on
plot(t,m23', '-b');
hold on
plot(t,m24','-m');
hold on
plot(t,m25', '-c');
hold on
plot(t,m26', '-k');
legend('m21','m22','m23','m24','m25','m26','Location','NorthEast')
```

```
126 ylabel('Angle Axis [ ] - Real','fontsize',14)
127 xlabel('Time [ms]','fontsize',14)
128  title ('Right Leg - Real Arnie','fontsize',14)
129 grid on
130 %
```

Bibliography

[1] J. Fellmann, *Diplomarbeit Entwicklung eines statisch stabilen Laufalgorithmus für einen zweibeinigen Laufroboter.* Hochschule Mannheim, 2007.

[2] Dynamixel rx-28 user's manual. ROBOTIS CO.,LTD.

[3] R. P. Goebel, *ROS by Example . A Do-it-Yourself Guide to the Robot Operating System*, 2012.

[4] E. F. Aaron Martinez, *Learning ROS for Robotics Programming*, P. P. Ltd, Ed., 2013.