Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Exploración de la aplicación V-REP
Exploration of software V-REP

Autor: Jesús de Miguel Fernández
Tutor: José Ángel Acosta Rodríguez

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Exploración de la aplicación V-REP
Exploration of software V-REP

Autor:

Jesús de Miguel Fernández

Tutor:

José Ángel Acosta Rodríguez

Profesor titular

Dep. de Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Exploración de la aplicación V-REP Exploration of software V-REP

Autor: Jesús de Miguel Fernández

Tutor: José Ángel Acosta Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia

A mis amigos

ACKNOWLEDGEMENTS

Me gustaría agradecer en primer lugar la labor de mi tutor por haberme guiado a lo largo de este camino con tantas bajadas y subidas.

Seguidamente, a mis padres y a mis tíos por la educación recibida sin la cual no estaría redactando este documento.

Finalmente, gracias a mi compañero Miguel Ángel, por haberme ayudado a convertir mi inglés en un inglés más formal.

ABSTRACT

The necessity of simulation software is esential in the robotic area. This software must be free, simple and powerful. Due to that urge, we are going to study in depth V-REP® software in order to discover his advantages and disadvantages. This software owns to Coppelia Robotics®, a new company which has obtained a remarkable popularity.

We are going to start with simple examples to familiarize us with V-REP®. With this preparation we are going to be ready to analyse a more complex case such as humanoid robot. In order to boost this software and know better how it works, from control's point of view, we are going to connect it with Matlab® via API.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

q	Joint Position	rad
qd	Joint Velocity	rad/s
qr	Joint Position Reference	rad
qdr	Joint Velocity Reference	rad/s
xr	XYZ Position Reference	m
Kp	Controller Proportional Term	$\frac{\text{controller output signal units}}{\text{error signal units}}$
Ti	Controller Integral Time	s
Td	Controller Derivative Time	s
Tau	Motor Torque	N·m
FK	Forward Kinematics	
J	Jacobian	
Pseudo	Jacobian Pseudoinverse	
DOF	Degrees of freedom	

1 INTRODUCTION

1.1 Why V-REP®?

Simulation software is essential to test any robotic system, in order to prevent it from hurting itself. Futhermore, it is really useful for education, due to the fact that the robot is not physically needed. The majority of softwares that can be found are designed only for robots, which have been designed by the company that has developed the software, and it is not common for them to be free or open source. Due to these arguments, we appeal to V-REP®. This Coppelia Robotics' software is a free simulator wich allows different simulation modes and includes algorithms, which are attractive for the calculus of essential elements in a robot such as inverse kinematics or collision detection. The graphical environment allows designing your own robot with primitive shapes or importing your own design from any design software. The user is totally free in their design.

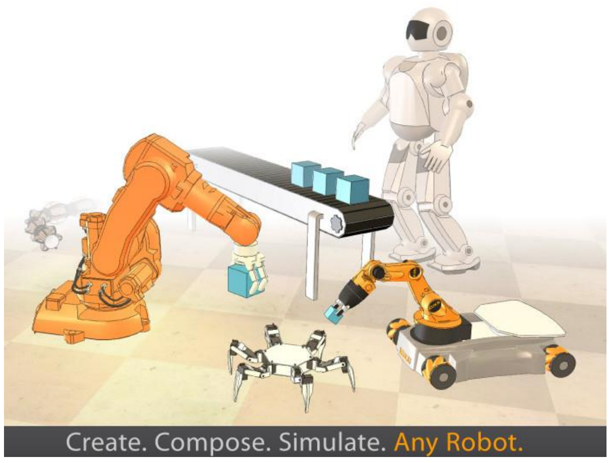


Figure 1-1.V-REP environment

1.2 Is V-REP easy to handle?

The developper gives you a lengthy manual in digital format in which the mayority of topics are detailed in order to familiarize with this software. Nevertheless, in some aspects, this manual is not clear enough and there are some aspects, which are not mentioned explicitly. From my own experience, it is rather difficult to get to know this environment, especially when you have to design your own robot. The importance of hierarchy tree is essential in this way. What is more, as I have mentioned before, there are some elements that you have to investigate by yourself, because they are not even explained in the user's manual. On the other hand, due to the attractiveness of the software, which is gainnig a remarkable importance and relevance, a large community has been created. Especifically, the own company has a forum in which you can discuss any of your doubts or problems and the own developers will answer you in a short period. It is a really useful tool, because the mayority of problems you can have in your travel with V-REP® are solved there.

1.3 Does V-REP have any limitations?

It is a simulator of significant interest. Although, it is true that there are some elements which are not clear enough and It is possible to not really know what you are modifyind in first instance. Especifically, the programming language (LUA) and control are very limitating. LUA is a very high-level language with predesigned functions which might make user feel hands tied. It presents two diffetents types of kinematics controlling: Pseudoinverse and DLS. Inside these structures, you can modify actually a few elements and sometimes leaves much to be desired. Related to dynamic control, it presents a large range of options that allows the user to implement his own controller code in LUA. Across this thesis, we are going to work with these control structures in a more detailed way, knowing what this software does, and how.

1.4 Analysis with V-REP

Our study with V-REP® will start with a simple example such as a 2 grades of freedom robot that only moves in the XY plane to familiarize with this environment. Then, we are going to study a more complex case such as humanoid robot, specifically, *Biolid Premium Type A* model.

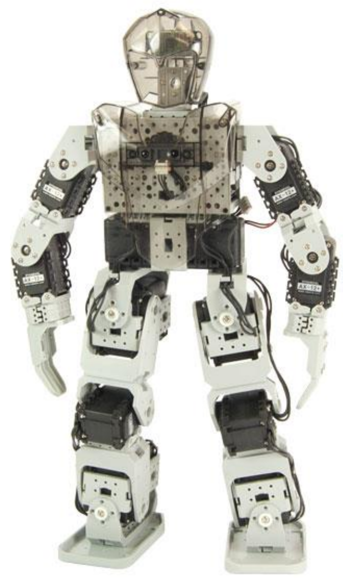


Figure 1-2.Biolid Premium Type A model

Because of the limitations that will be found, and to try to understand in a better way the control structures that this software uses, we look for a conexión *API* with MATLAB® in which V-REP® is the server an MATLAB® the client.

2 2 DOF PLANAR ROBOT IN V-REP®

In this chapter, we are going to explore this new software with an easy example such as two grades of freedom XY planar robot. Only the tools that V-REP has are going to be used in order to build it up. The steps that are going to be described are developed in the scenes and models chapter of the user's manual (Bibliography [8]) for other configuration.However, it seems to me that there are some points I am going to develop which are not clear enough.

2.1 Building up

The final result can be seen in Figure 2-1.

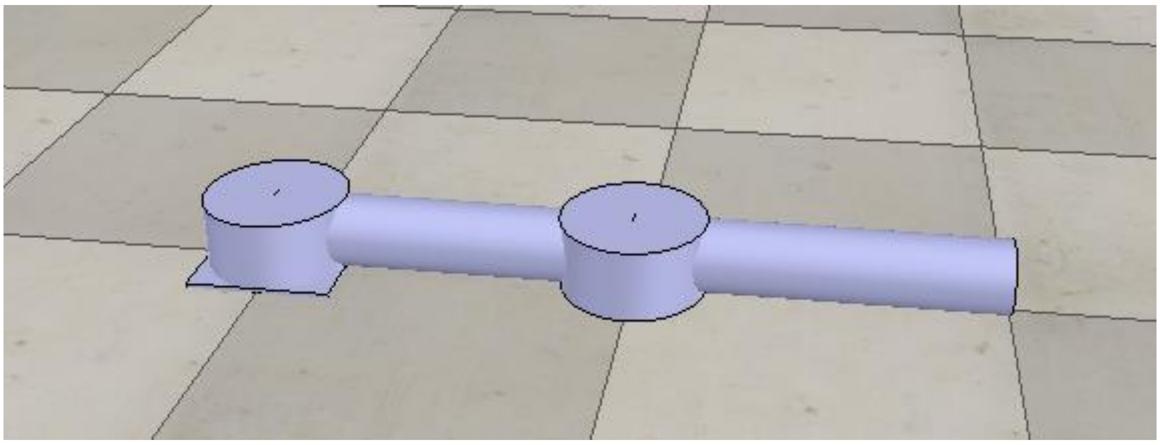
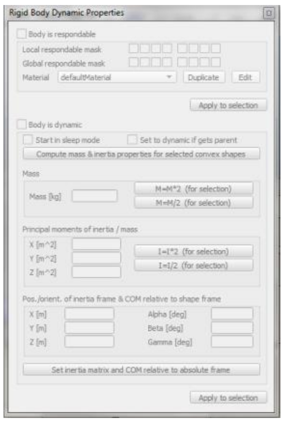


Figure 2-1.2 DOF robot in V-REP

2.1.1 Links

As it can be seen, it is composed just by cylinders and a little cube wich is the base. To create these shapes I select the pure shape I want (cilinder, sphere or cube). Once it is created, I can situate and orientate it as I want. It has to differentiate between two natures for the same link in its properties. It can be seen in figures 2-2 and 2-3.

- **Non-dynamically enabled link:** This is the original shape of the link. This nature is necessary to collision calculus and object detection by vision among other functions. We configurate the shape properties like that:



Rigid Body Dynamic Properties

☐ Body is respondable

Local respondable mask ☐☐☐☐ ☐☐☐☐

Global respondable mask ☐☐☐☐ ☐☐☐☐

Material: defaultMaterial

☐ Body is dynamic

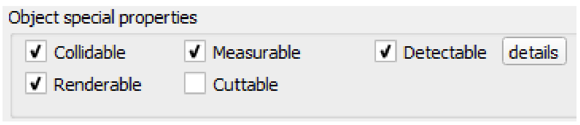
☐ Start in sleep mode ☐ Set to dynamic if gets parent

Mass [kg]:

Principal moments of inertia / mass: X [m^2], Y [m^2], Z [m^2]

Pos./orient. of inertia frame & COM relative to shape frame: X [m], Y [m], Z [m], Alpha [deg], Beta [deg], Gamma [deg]

(a)



Object special properties

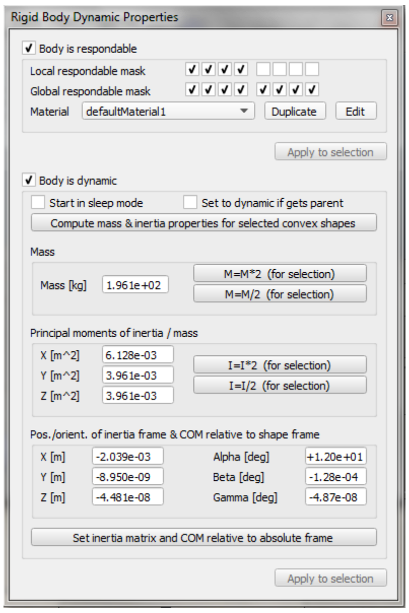
✓ Collidable ✓ Measurable ✓ Detectable

✓ Renderable ☐ Cuttable

(b)

Figure 2-2.Non-dynamically enabled link body properties (a) and common properties (b).

- **Dynamically enabled link:** This nature looks for a simplified version link to help dynamic module and to enjoy a better simulation experience. Here the link presents: mass, inertia and motor torque response. We configurate the shape properties as it follows:



Rigid Body Dynamic Properties

[✓] Body is respondable

Local respondable mask [✓] [✓] [✓] [✓] [ ] [ ] [ ] [ ]

Global respondable mask [✓] [✓] [✓] [✓] [✓] [✓] [✓] [✓]

Material: defaultMaterial1

[✓] Body is dynamic

[ ] Start in sleep mode [ ] Set to dynamic if gets parent

Mass

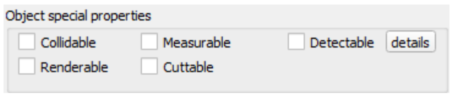
Mass [kg]	1.961e+02

Principal moments of inertia / mass

X [m^2]	6.128e-03
Y [m^2]	3.961e-03
Z [m^2]	3.961e-03

Pos./orient. of inertia frame & COM relative to shape frame

X [m]	-2.039e-03	Alpha [deg]	+1.20e+01
Y [m]	-8.950e-09	Beta [deg]	-1.28e-04
Z [m]	-4.481e-08	Gamma [deg]	-4.87e-08



Object special properties

[ ] Collidable [ ] Measurable [ ] Detectable

[ ] Renderable [ ] Cuttable

(a) (b)

Figure 2-3.Dynamically enabled link body properties (a) and common properties (b)

It is going to be described each of the options which appear in the object special properties dialog:

- **Collidable**: allows enabling or disabling collision detection capability for the selected collidable object.

- **Measurable**: allows enabling or disabling minimum distance calculation capability for the selected measurable object.

- **Detectable**: allows enabling or disabling proximity sensor detection capability for the selected detectable object.

- **Renderable**: allows enabling or disabling the vision sensor detection capability for the selected renderable object.

- **Cuttable**: allows enabling or disabling the mill cutting capability for the selected cuttable object.

As we will see next in hierarchy tree, non-dynamic links depends on dynamic ones and they move with them. Non-dynamic links are only like shell. The difference I am talking about can be found in figure 2-4.

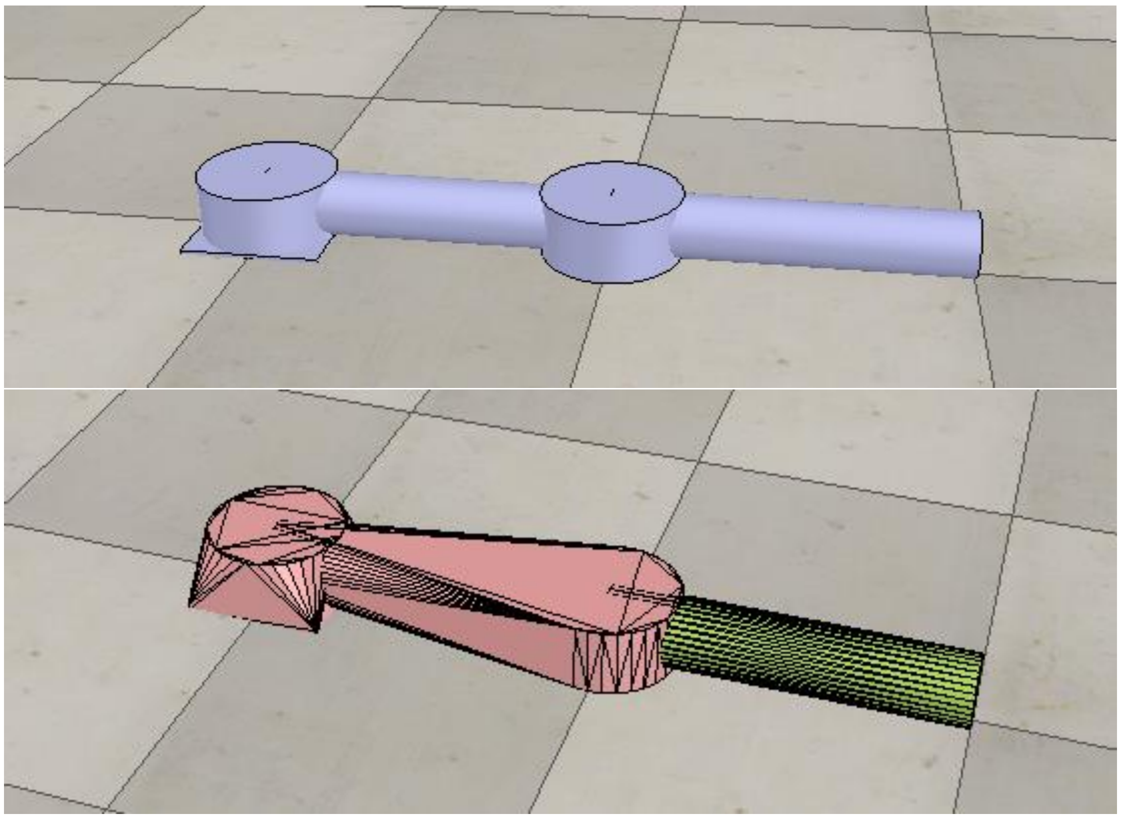


Figure 2-4.Non-Dynamically enabled links robot (above) and dynamically enabled link robot (down)

It can be seen that dynamic shapes are much simpler than non-dynamic shapes. To have this result we need to select he shape and click on *Edit/Morph selection into convex shapes* option. During the simulation, we are only interested in seeing non-dynamically enabled links behavior, because of that we need to change the mask of dynamically enabled links inside *Scene Object Properties/Common/Camera Visibility layers*. To determine in which layer we put our links we need to know which are activated in the scene inside *Tools/Layers*.

2.1.2 Joints

In the same way we have added shapes, joints can be added (revolute, prismatic and spherical). There are two posibilities for situating and orientating joints in our model:

- **Manually:** as we did with links.
- **Deviant-Hatemberg module:** You introduce all parameters and they will be situated and orientated automatically.

The different modes to use a joint can be found in figure 2-5.

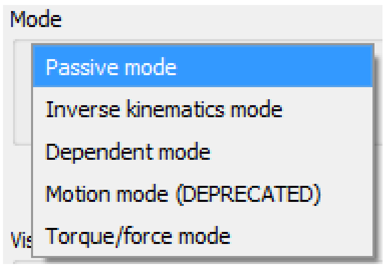
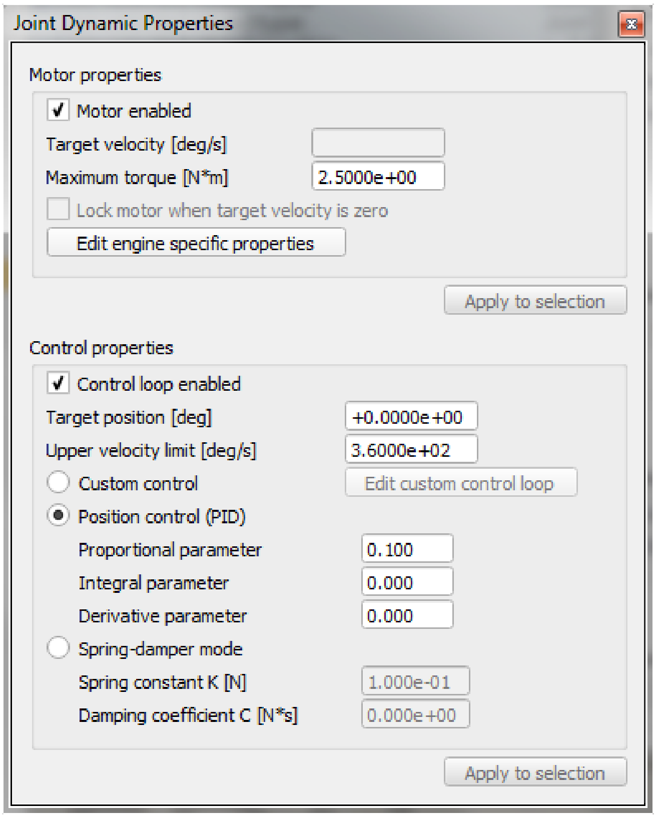


Figure 2-5.Joint modes

We are going to focus on *torque/force mode*, to dynamic control, and *inverse kinematic mode*, to obtain jacobian matrix, follow paths and kinematics control. Besides, we must habilitate: motors, dynamic control and set maximum velocity and torque in the joint dynamic properties as we can see in figure 2-6.


Joint Dynamic Properties

Motor properties

[✓] Motor enabled

Target velocity [deg/s]	
Maximum torque [N*m]	2.5000e+00

[ ] Lock motor when target velocity is zero

Control properties

[✓] Control loop enabled

Target position [deg]	+0.0000e+00
Upper velocity limit [deg/s]	3.6000e+02

○ Custom control

● Position control (PID)

Proportional parameter	0.100
Integral parameter	0.000
Derivative parameter	0.000

○ Spring-damper mode

Spring constant K [N]	1.000e-01
Damping coefficient C [N*s]	0.000e+00

Figure 2-6.Joint dynamic properties

We are going to study soon what each element means.

2.1.3 Hierarchy

This is the final and most important step for our model to work properly. I would like to say that in the manual I have mentioned before this topic does not have the relevance that it deserves. Firstly, we have to set a base and then create a dependency with it. It is important to know that a joint moves what depends on it, but it has to have only ONE body directly linked. If we linked a second one, it would not notice motor's presence and it is linked with the last one neither so it would fall. To combine two links that depend on the same joint, I have chosen *Merge selected shapes* option in *Edit/Grouping/Merging*. The result is a new figure which includes the two links. Our model's hierarchy, taking into account what has been said, can be found in figure 2-7.

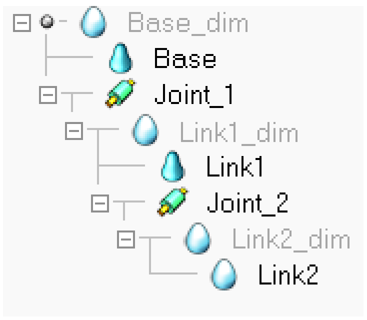


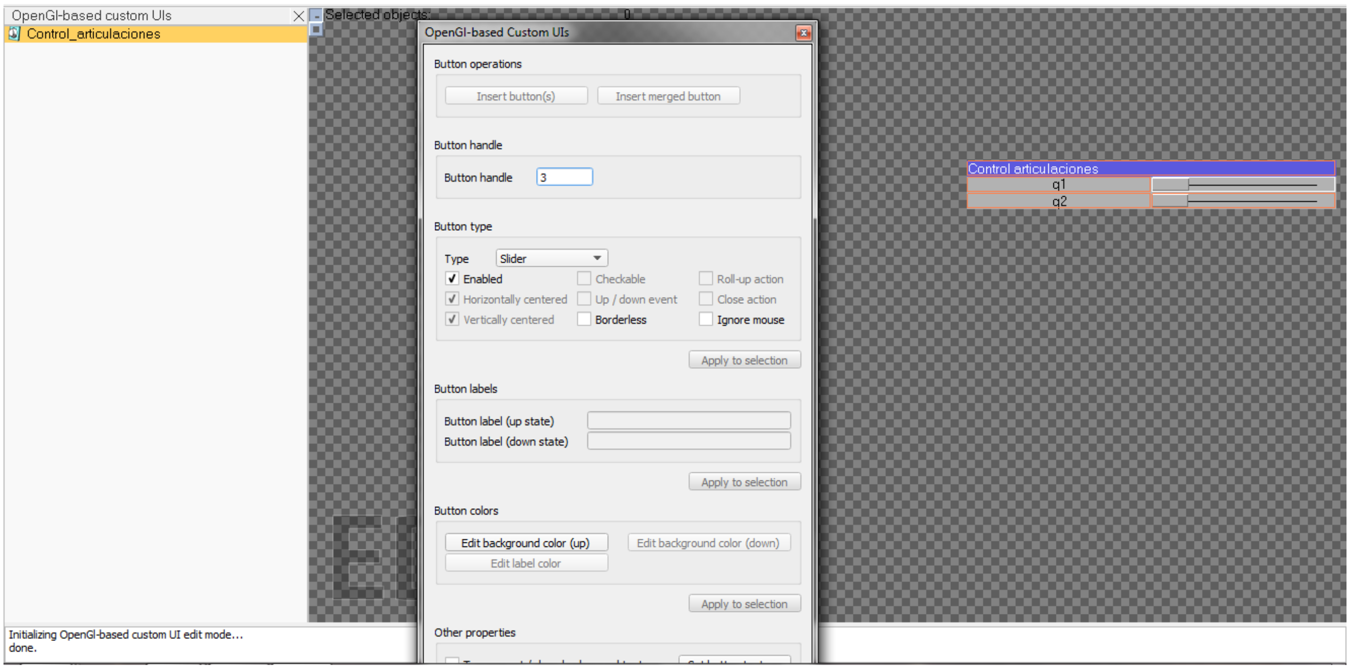
Figure 2-7.Model's hierarchy tree

2.2 Dynamic control

At this point we are going to set joints values by user interface composed by sliders that can be added in the software. For joint control we are going to leave the default parameters, a proporcional controller with gain 0, 1 [(rad/s)/rad].

2.2.1 Slider interface

As I have said before, we are going to design it with a tool that V-REP brings. Every slider has a position in the interface (*button handle*) and it will be useful at programming time. It can be seen in figure 2-8.



Button handle: 3

Type: Slider

[✓] Enabled

[✓] Horizontally centered

[✓] Vertically centered

Initializing OpenGl-based custom UI edit mode...
done.

Figure 2-8.Designing user interface

It is interesting to say that this functionality is not brought by new versions of V-REP. I have written this scene with V-REP version 3.04.00 (rev 1) and when I updated software to a newer version, user interface appears only during simulation and works properly, but can not be modified.

2.2.2 Code

The structure of my code must follow in that case and the different types are found in the user manual. Furthermore, all predefined API functions language brings are also explained in it. It would make no sense to detail which parameter receive each one when it is said in the manual. Script model, which is more adequate to our application, is non-threaned child script. I link this script to my robot base as it can be seen in figure 2-9.

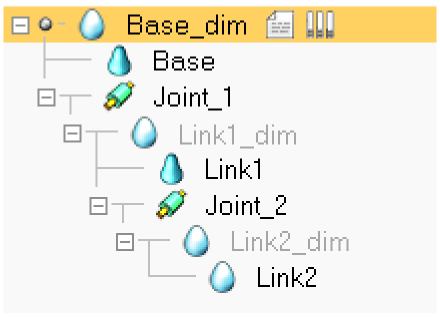


Figure 2-9.Script linked to robot base

In this script, which can be found in appendix A clearly expalined, we can see two different parts: variables initialization and code execution during simulation.

2.2.3 Demo

All this points are properly implemented in *2GOF_dynamicslider.ttt* scene.

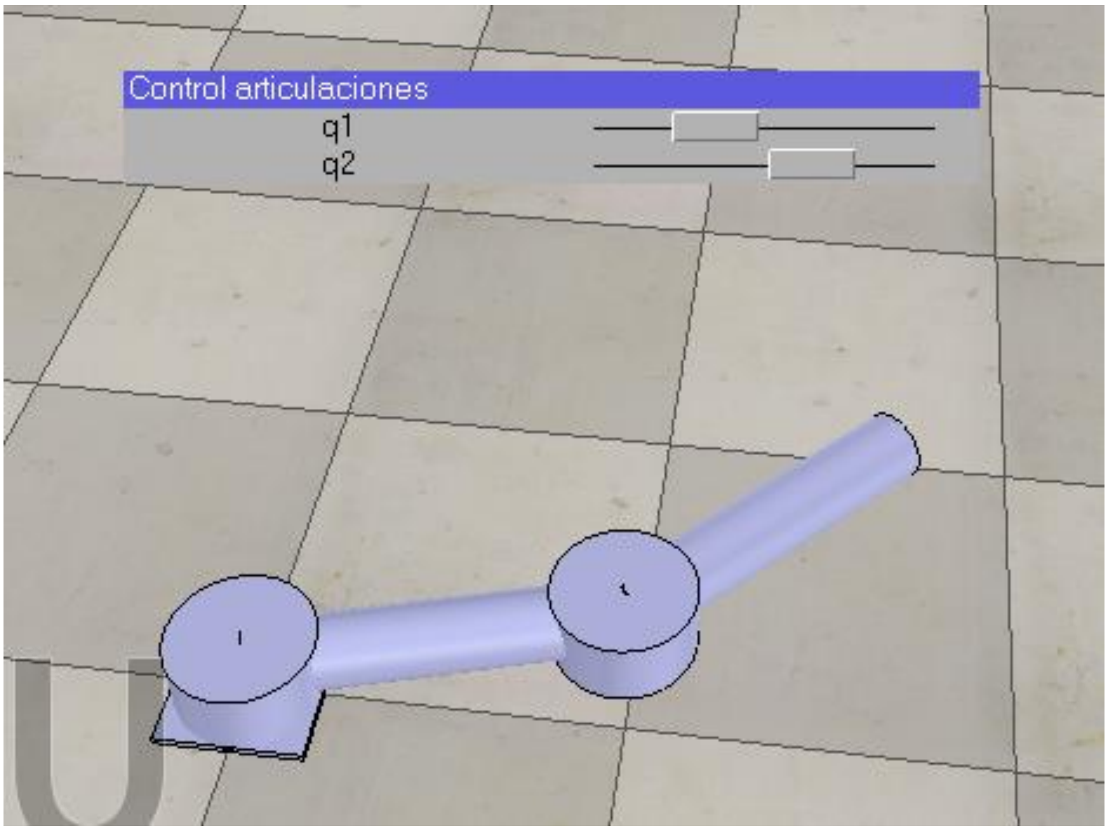


Figure 2-10.Dynamic control with slider demo

2.2.4 How do we control?

Firstly, it is not clear enough how from moving the slider we have obtained the desired position, from control's point of view.

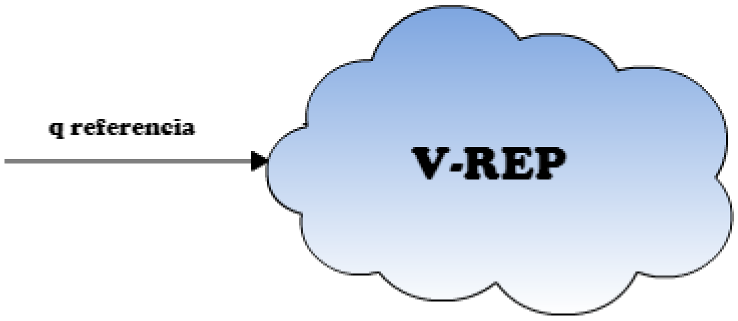


Figure 2-11.Action from user's point of view

As we are going to implent in Matlab afterwards, it is interesting to know what all parameters in the joint dynamic properties mean inside the control loop. Thinking over the behavior of each parameter we arrive to the control diagram which can be seen in figure 2-12.

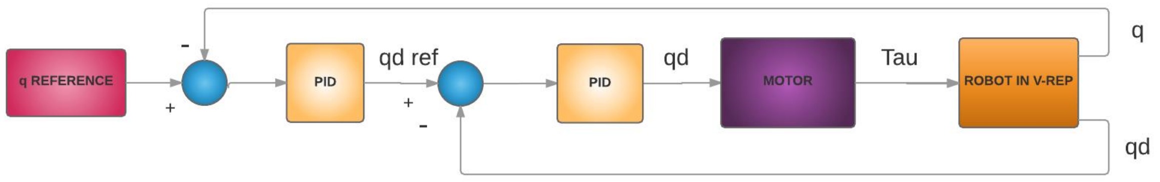
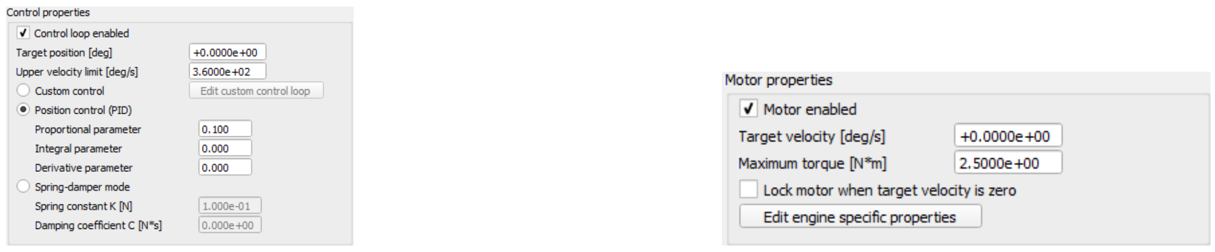


Figure 2-12.Dynamic control diagram

From left to right, we can distinguish all the elements of this diagram in the options as we can see in the figure 2-13.



Control properties

- [✓] Control loop enabled
- Target position [deg]: +0.0000e+00
- Upper velocity limit [deg/s]: 3.6000e+02
- ○ Custom control — Edit custom control loop
- ● Position control (PID)
  - Proportional parameter: 0.100
  - Integral parameter: 0.000
  - Derivative parameter: 0.000
- ○ Spring-damper mode
  - Spring constant K [N]: 1.000e-01
  - Damping coefficient C [N*s]: 0.000e+00

Motor properties

- [✓] Motor enabled
- Target velocity [deg/s]: +0.0000e+00
- Maximum torque [N*m]: 2.5000e+00
- [ ] Lock motor when target velocity is zero
- Edit engine specific properties

(a) First controller

(b) Second controller and motor

Figure 2-13.Assignment in the dynamic control diagram

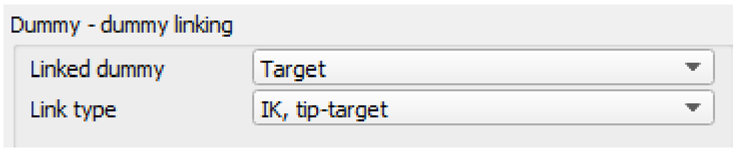
In the first controller, if we enable *Position control (PID)*, we will be controlling in position while we modulate in velocity with PID that brings implementated. In the next dialog, *Motor enable* option is to enable or disable motor block. Target velocity option will appear inaccessible while we had been enabled position controller. In case we do not enable position control, a joint velocity reference will be passed which will be reached inmediatly if the maximum torque is big enough, in other case, it will gradually reach it.

2.3 Kinematics control

Our goal now is to get jacobian for a determinated position. By defining differents position or a path, we will gain different values of the jacobian which can be interpolated to obtain symbolic jacobian. We have to change joint mode to *Inverse kinematics mode* to work properly in this case.

2.3.1 Obtaining position in the target location

To create a new position we have to create two dummies, one that will be in the extreme of the robot (tip) and other one in the location we wish (target). Inverse kinematics module, which will be explained in the next point, will generate the correspondent position for that location and orientation in the plane. We have to configurate the relation between the two dummies next as it can be seen in figure 2-14.

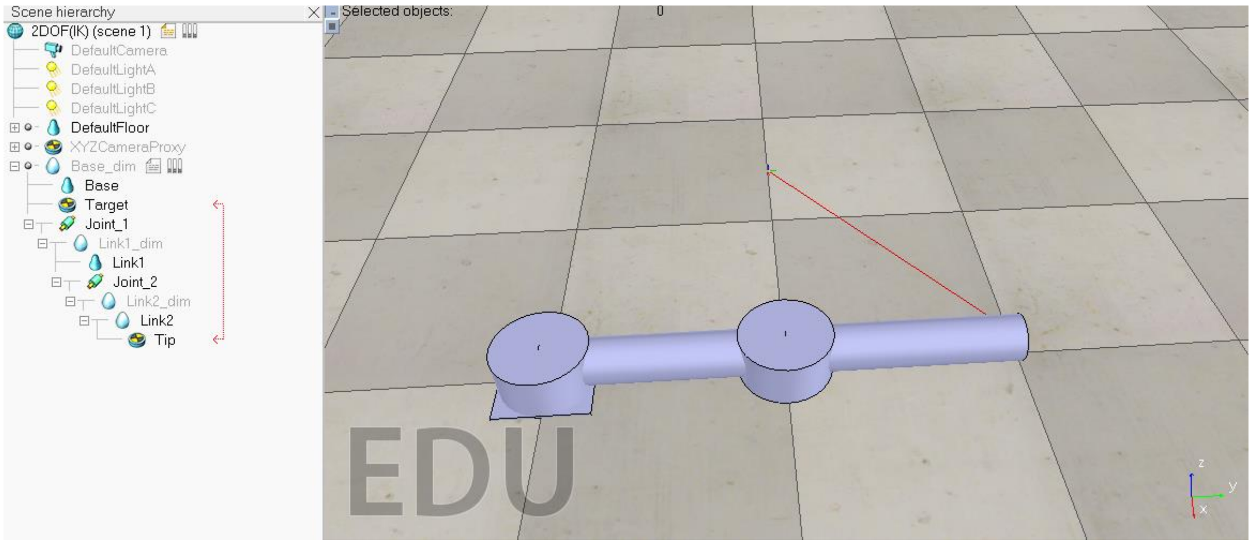

Dummy - dummy linking

Linked dummy: Target

Link type: IK, tip-target

Figure 2-14.Configuration between dummies

In the hierarchy tree, they must be distributed like it can be seen in figure 2-15.


Scene hierarchy

```
2DOF(IK) (scene 1)
├── DefaultCamera
├── DefaultLightA
├── DefaultLightB
├── DefaultLightC
├── DefaultFloor
├── XYZCameraProxy
└── Base_dim
    ├── Base
    ├── Target
    └── Joint_1
        └── Link1_dim
            ├── Link1
            └── Joint_2
                └── Link2_dim
                    └── Link2
                        └── Tip
```

Figure 2-15.Correct distribution of dummies in hierarchy

2.3.2 Inverse Kinematics Module

With this module, we can create a kinematics controller for the IK group we wish two dummies in our case. In the options of the module, we can see different options, of which we need to know their meaning.

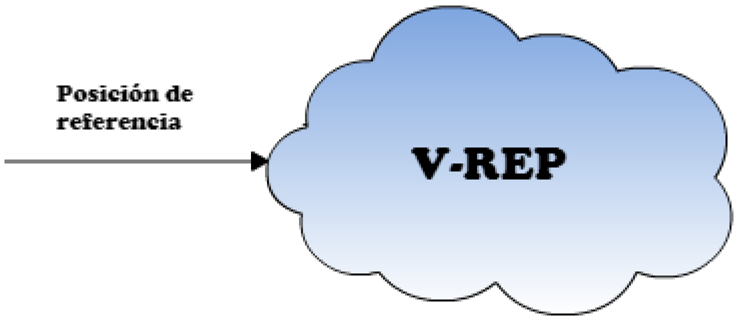
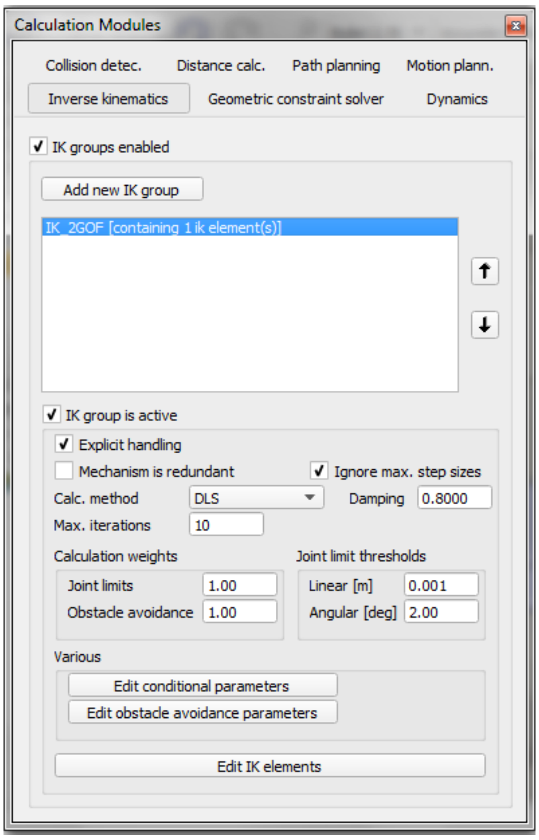


Figure 2-16.Action from user's point of view

We can choose the method of calculus between Pseudoinverse and DLS. DLS is helpful when Pseudoinverse fails, for example, when a robot is near a singularity o it is a redundant robot. In case we select DLS, we can choose what damping we want. The bigger the damping is, the better approximation will be, but with the inconvenient of it being slower. Related to the number of iterations, it depends on the method we have choosen and the resolution we want. If we choose DLS, a bigger number of iterations will be necessary. For our example, we will choose a DLS with moderate damping as we can see in figure 2-17.



Calculation Modules

Collision detec. | Distance calc. | Path planning | Motion plann. | Inverse kinematics | Geometric constraint solver | Dynamics

✓ IK groups enabled

Add new IK group

IK_2GOF [containing 1 ik element(s)]

✓ IK group is active

✓ Explicit handling

Mechanism is redundant

✓ Ignore max. step sizes

Calc. method: DLS

Damping: 0.8000

Max. iterations: 10

Calculation weights

Joint limits: 1.00

Obstacle avoidance: 1.00

Joint limit thresholds

Linear [m]: 0.001

Angular [deg]: 2.00

Various

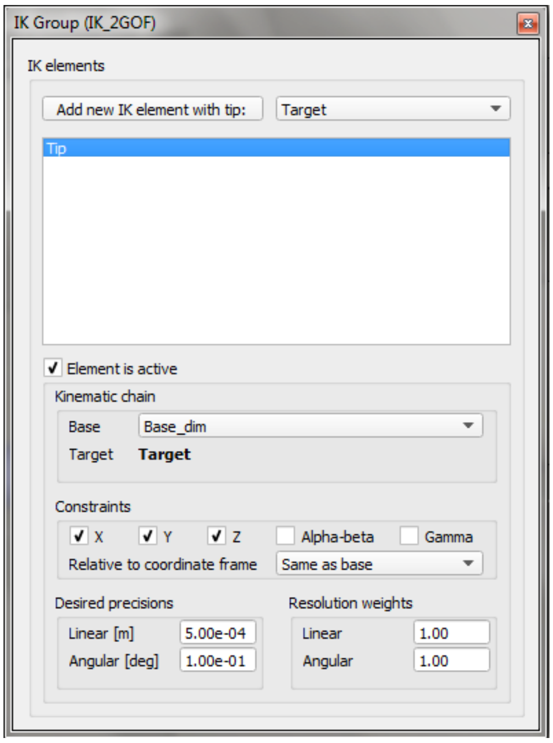
Edit conditional parameters

Edit obstacle avoidance parameters

Edit IK elements

Figure 2-17.IK module configuration

In the next step, we have to create an IK group for our two dummies which has to have the configuration we can see in figure 2-18.



IK Group (IK_2GOF)

IK elements

Add new IK element with tip: Target

Tip

✓ Element is active

Kinematic chain

Base: Base_dim

Target: **Target**

Constraints

✓ X ✓ Y ✓ Z Alpha-beta Gamma

Relative to coordinate frame: Same as base

Desired precisions

Linear [m]: 5.00e-04

Angular [deg]: 1.00e-01

Resolution weights

Linear: 1.00

Angular: 1.00

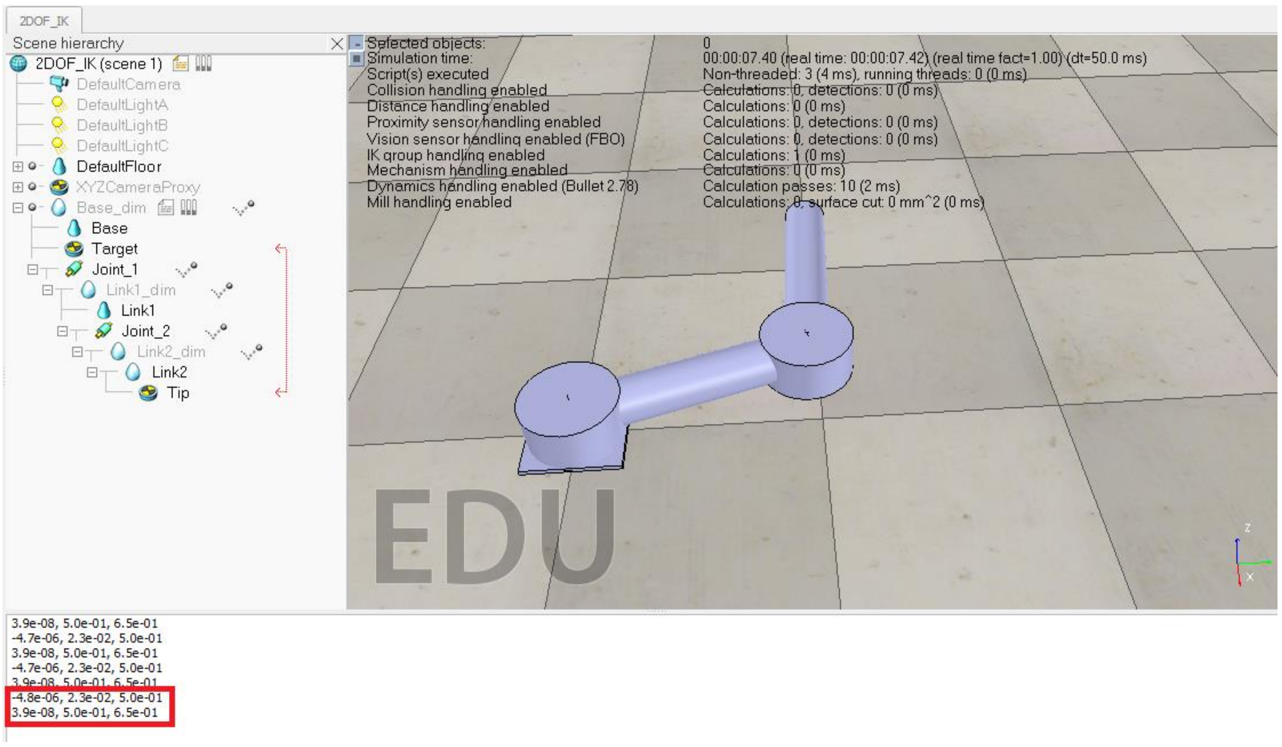
Figure 2-18.IK group configuration

2.3.3 Code

It is necessary to clearly know how functions retrieve data in this case to interpretate them correctly. Clear examples of that are functions which get Jacobian matrix, Deep awareness of their outputs is needed.This code can be found in the appendix B clearly commented and detailed.

2.3.4 Demo

All this points are correctly implemented in *2GOF_IK.ttt* scene. It can be seen who Jacobian matrix appears on the console window in figure 2-19.



```
Selected objects:                            0
Simulation time:                             00:00:07.40 (real time: 00:00:07.42) (real time fact=1.00) (dt=50.0 ms)
Script(s) executed                           Non-threaded: 3 (4 ms), running threads: 0 (0 ms)
Collision handling enabled                   Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                    Calculations: 0 (0 ms)
Proximity sensor handling enabled            Calculations: 0, detections: 0 (0 ms)
Vision sensor handling enabled (FBO)         Calculations: 0, detections: 0 (0 ms)
IK group handling enabled                    Calculations: 1 (0 ms)
Mechanism handling enabled                   Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)      Calculation passes: 10 (2 ms)
Mill handling enabled                        Calculations: 0, surface cut: 0 mm^2 (0 ms)
```

```
3.9e-08, 5.0e-01, 6.5e-01
-4.7e-06, 2.3e-02, 5.0e-01
3.9e-08, 5.0e-01, 6.5e-01
-4.7e-06, 2.3e-02, 5.0e-01
3.9e-08, 5.0e-01, 6.5e-01
-4.8e-06, 2.3e-02, 5.0e-01
3.9e-08, 5.0e-01, 6.5e-01
```

Figure 2-19.Jacobian result on the console window

2.3.5 How do we control?

IK module is really opaque and we do not really know what it does. It follows the control diagram which can be foun in figure 2-20.

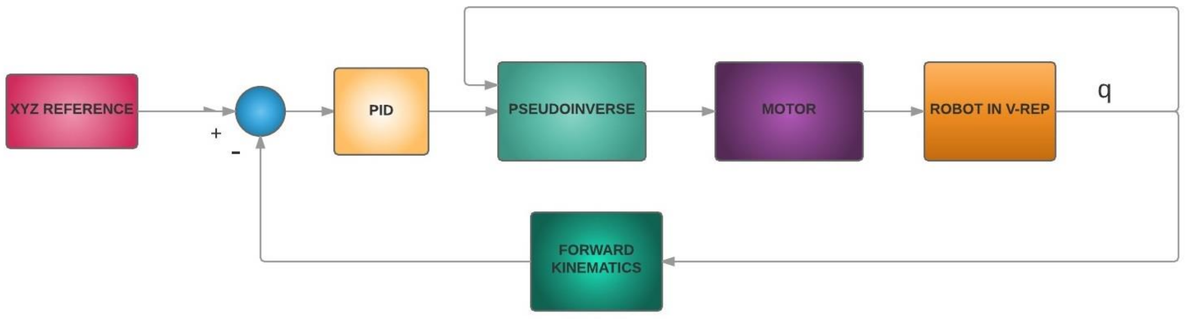


Figure 2-20.Kinematics control diagram

The reference we have passed to it is the position target dummy position. We have found an important limitation which is not being able to modify neither the controller nor the method of calculus. This can be poor in certain cases. We will explore this diagram in a deeper way in Matlab.

3 HUMANOID ROBOT IN V-REP®

In this section we are going to develop the example of a humanoid robot (Biolid Type A) which can be found in Ingeniería de Sistemas y Automática department of our school. It is interesting to focus on this configuration to set aside common configurations studied along degree.

3.1 Buildind up

The mayority of shapes I have used to build it up have been proportionated by Rafael Matínez Márquez in his thesis *Diseño de un sistema de vuelo de un robot humanoide*. It is detailed in the manual that the favorite format to import shapes from others design softwares, *CATIA V5* in my case, is *.stl* format. All shapes can be found in the folder named as *Piezas*. Final result can be seen in figure 3-1.

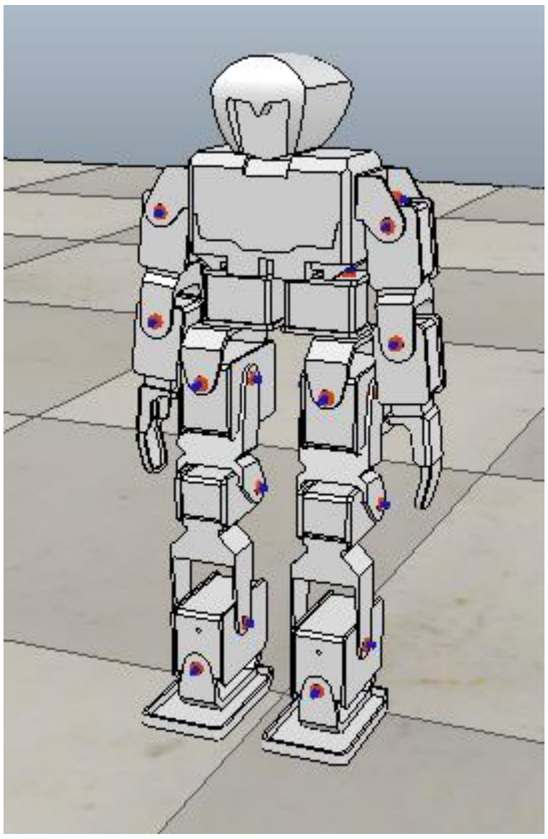
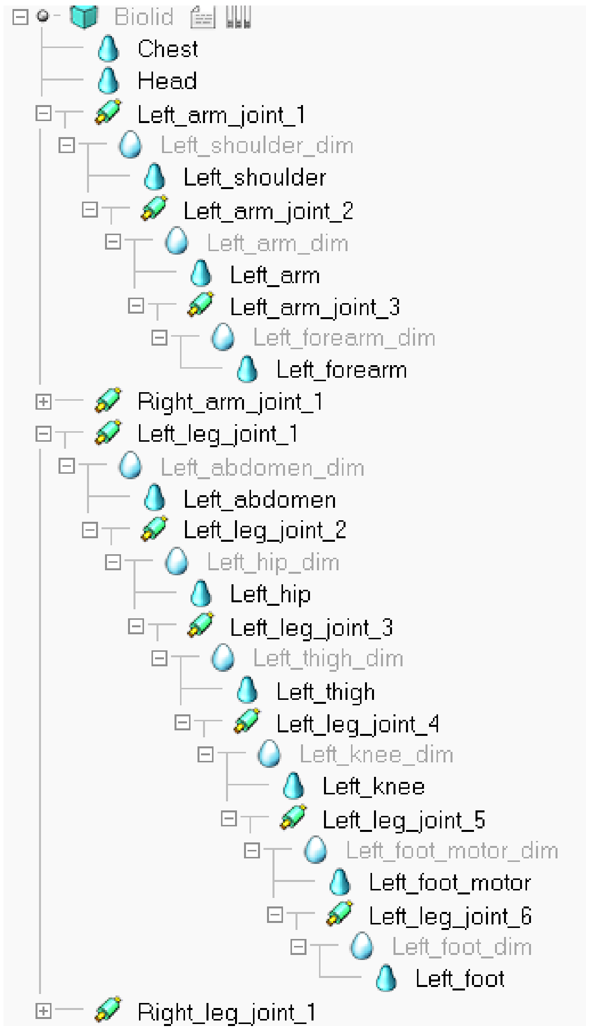


Figure 3-1.Humanoid robot in V-REP

It is needed to create non-dynamically enabled links and dynamically enabled links. We have to select the same options we have chosen in 2 DOF robot. Furthermore, we have to simplify dynamically enabled links with *Morph selection into convex shapes* option. But first, we need to connect all the links properly; in the same way in the same way it was done. I have been especially careful with elements hierarchy and shape merging. In this case I have choosen the chest as base of our robot and legs and arms elements dependent of it. Finally, hierarchy resulted follows the scheme we can see in figure 3-2.



```
Biolid
  Chest
  Head
  Left_arm_joint_1
    Left_shoulder_dim
      Left_shoulder
    Left_arm_joint_2
      Left_arm_dim
        Left_arm
      Left_arm_joint_3
        Left_forearm_dim
          Left_forearm
  Right_arm_joint_1
  Left_leg_joint_1
    Left_abdomen_dim
      Left_abdomen
    Left_leg_joint_2
      Left_hip_dim
        Left_hip
      Left_leg_joint_3
        Left_thigh_dim
          Left_thigh
        Left_leg_joint_4
          Left_knee_dim
            Left_knee
          Left_leg_joint_5
            Left_foot_motor_dim
              Left_foot_motor
            Left_leg_joint_6
              Left_foot_dim
                Left_foot
  Right_leg_joint_1
```

Figure 3-2.Humanoid hierarchy tree

It can be taken into account what I mentioned in the last chapter about merging shapes and not to have nested properly links with joints. A bad hierarchy model can produce results as that when it tries to move joints in the ankle as it can be seen in figure 3-3.

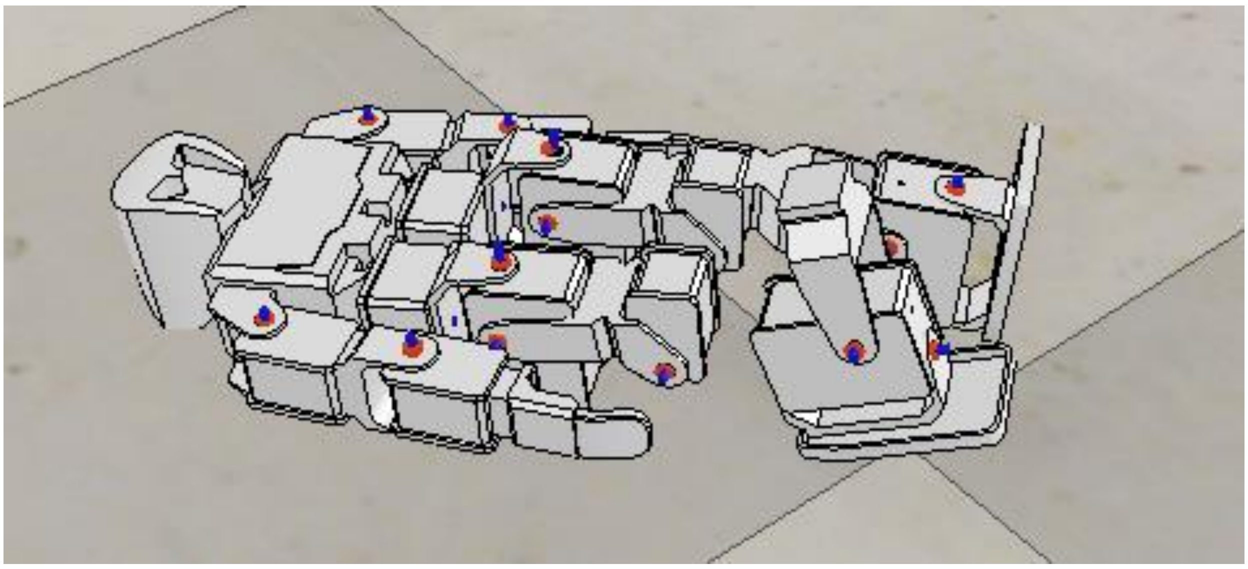


Figure 3-3.Result of a bad hierarchy three

Finally, we have to configurate dynamically enabled links in the way mentioned before to have a properly behavior of the robot during simulation as can be seen in Figure 3-4.

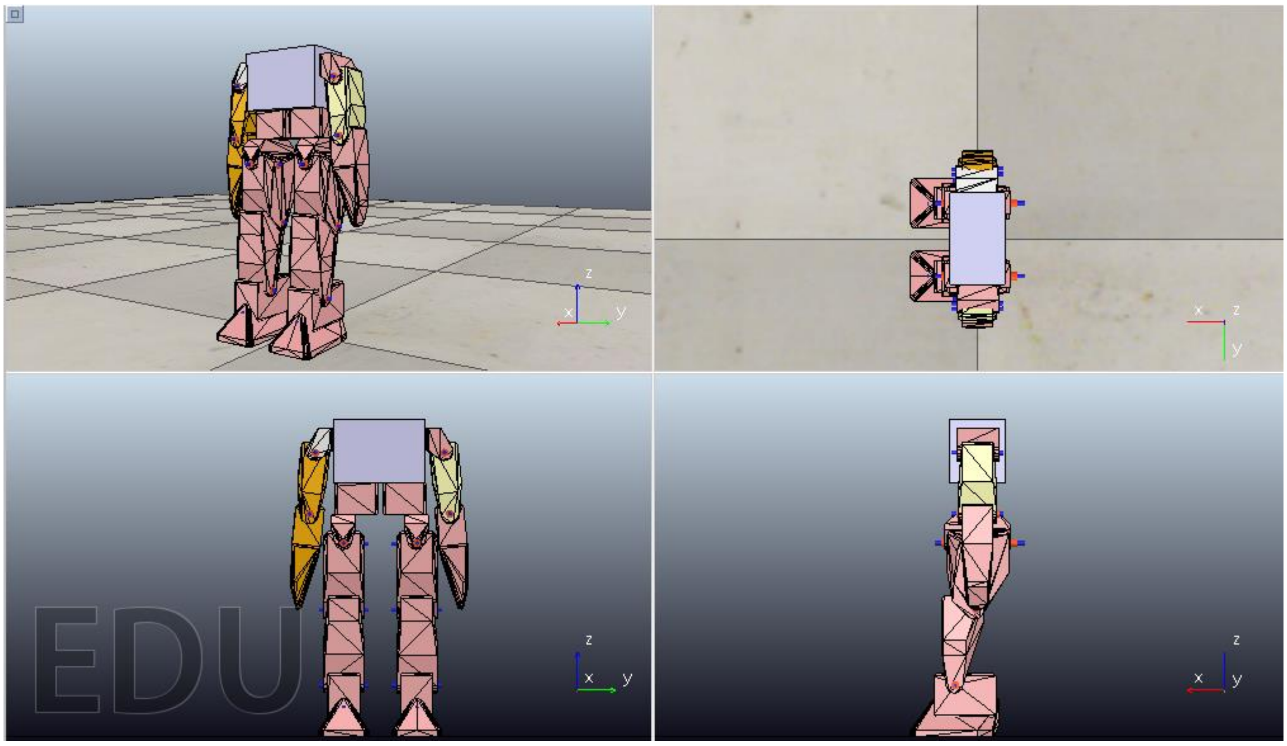
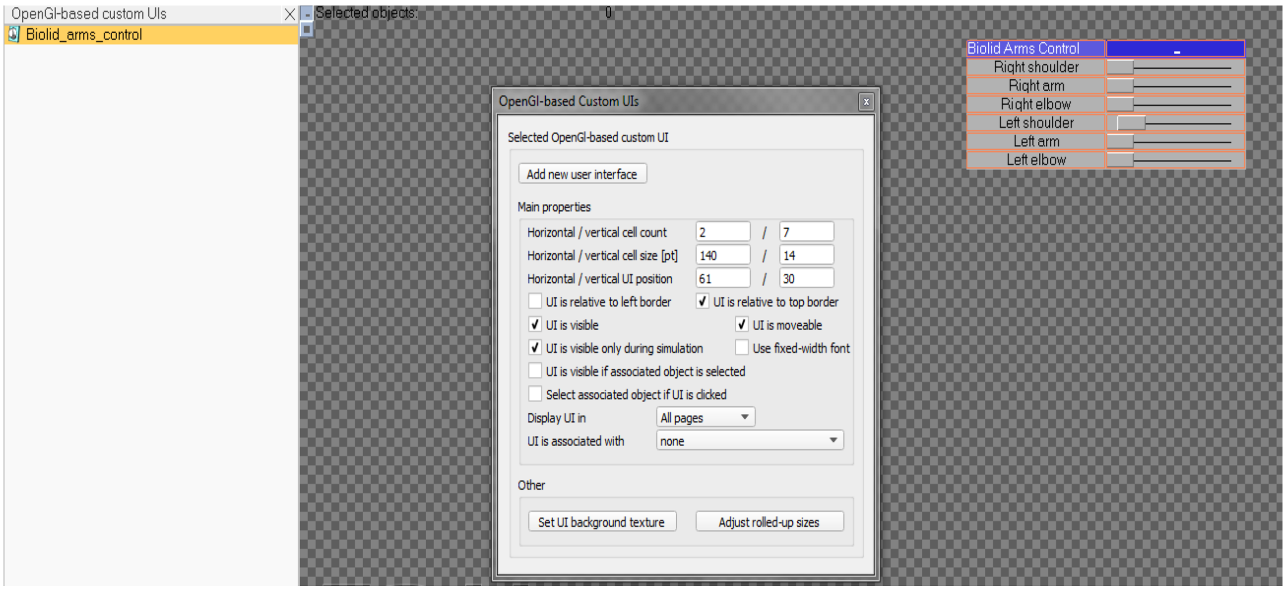


Figure 3-4.Humanoid dynamically enabled links

We cannot forget that in the simulation we are only interested in seeing non-dynamically enabled links, due to the fact that they are the ones that may collide with other elements.

3.2 Arms dynamic controller

To control dynamically both arms I am going to proceed like I did in the second chapter. Firstly, I select torque mode for joints and leave all default parameters. Afterward, I add the same type of user interface I have created before as we can see in figure 3-5.



Selected OpenGl-based custom UI

Main properties

Horizontal / vertical cell count	2	7
Horizontal / vertical cell size [pt]	140	14
Horizontal / vertical UI position	61	30

☐ UI is relative to left border ✓ UI is relative to top border
✓ UI is visible ✓ UI is moveable
✓ UI is visible only during simulation ☐ Use fixed-width font
☐ UI is visible if associated object is selected
☐ Select associated object if UI is clicked

Display UI in: All pages
UI is associated with: none

Other

Figure 3-5.UI to arms control

3.2.1 Code

Conceptually, it is not different from the case of two grades of freedom. This code can be found in the appendix C clearly commented and detailed.

3.2.2 Demo

All has been properly implemented in *Arms_control.ttt* scene.

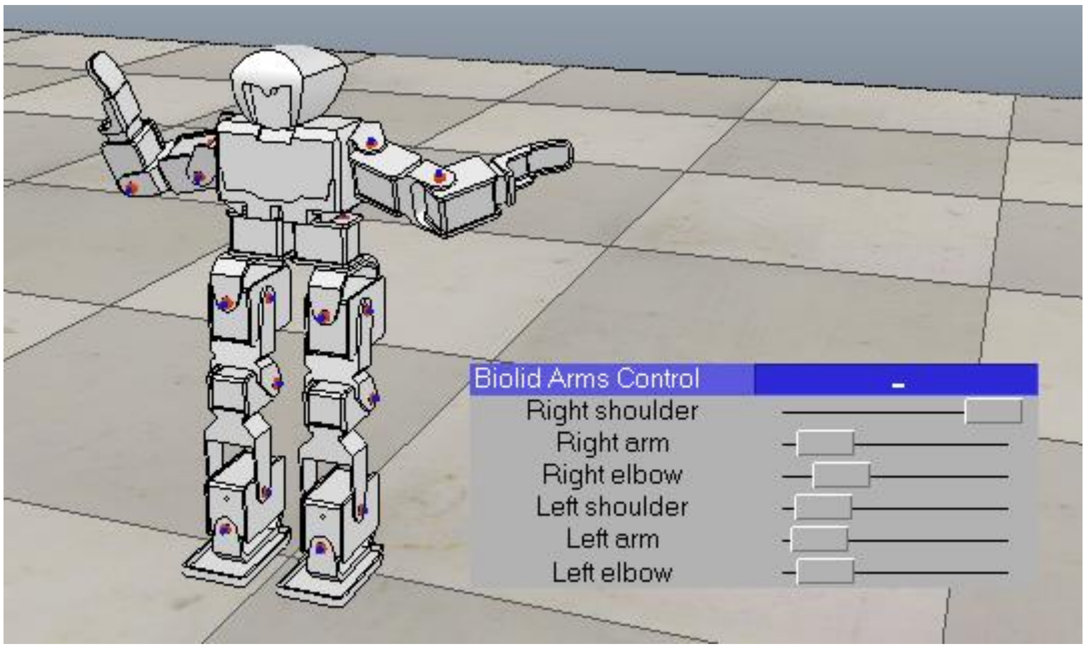


Figure 3-6.Arms control demo

3.3 Left Leg dynamic controller

We can control each leg in the same way now, for this example, the left one has been chosen. Obviously, as I will handle it as I wish, my robot will lose its balance and it will fall. This can be found correctly implemented in *Leg_control.ttt* scene and this code can be found in the appendix D clearly commented and detailed.

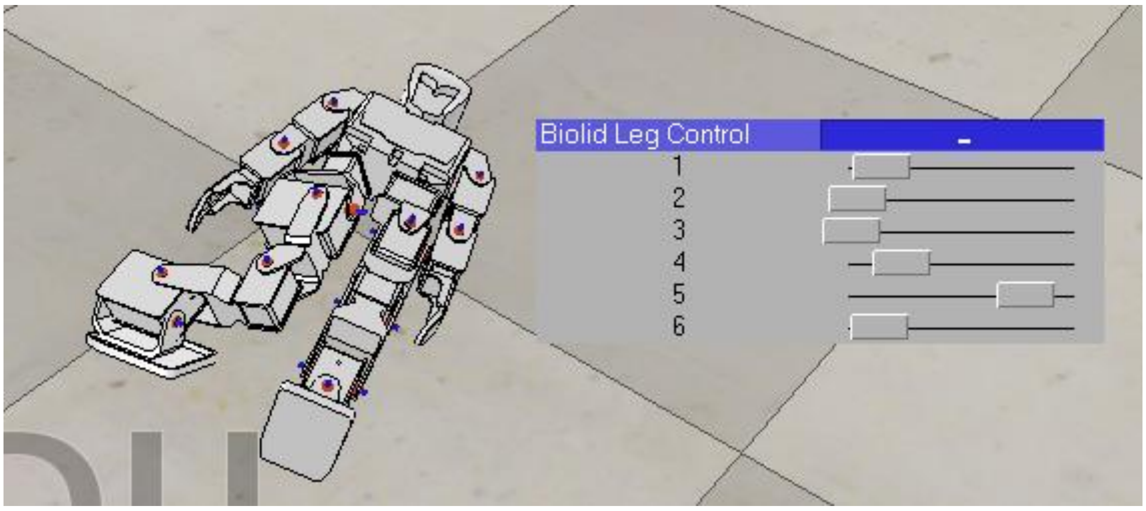


Figure 3-7.Leg control demo

3.4 Controlling in open loop to walk

It is not possible to directly modify kinematics control limitations that will be present in this point. As we cannot control stability elements such as waist and legs, we are going to aproach the problem from two different ways.

3.4.1 Kinematics control: IK chain with dummies

The main idea is to establish two kinematics chains for each foot. Each chain has associated two dummies: a tip (situaded in the foot sole) and a target (situated in the end of the foot). I have created two kinematics chains which include all the joints from each leg. Each chain will follow a straight path with a determined velocity that can be modify. Each kinematics chain will move following the dummies' movement. Along code execution we will be gainging robot absolute position and dummie in the path to determinate the next step. This idea is discussed in certains topics in the forum I have mentioned in the introduction. Furthermore, it is implemented correctly in an example V-REP brings (*Asti robot*).In this example, it is created an accurate path so that it exists a certain swinging that looks like it is a closed loop control when it is not. For our robot waist does not represent the main support, it uses the inclination of ankles and this method does not take into account that. From the example mentioned, it can be extracted that a really specifically path must been created, which does not appear quite interesting from control's point of view. Due to these arguments, we ruled out this method. Nevertheless, this code can be found in the appendix E clearly commented and detailed. It can be found implemented in *Biolid_walk_ik.ttt* scene.

3.4.2 Dynamic control: State machine

In this case, I am going to design a little state machine where in each state a different position of the step is declarated. Changing between states is conditionated by certain joint arrives to established refence. Different positions that are going to be established can be seen in figures 3-8, 3-9 and 3-10.

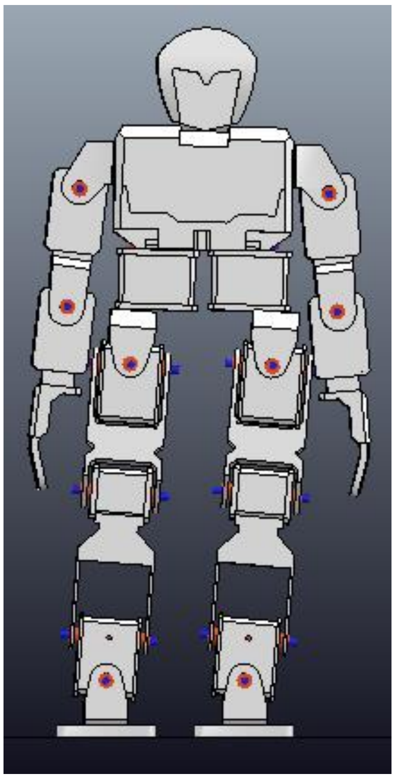


Figure 3-8.Required inclination to not lose balance

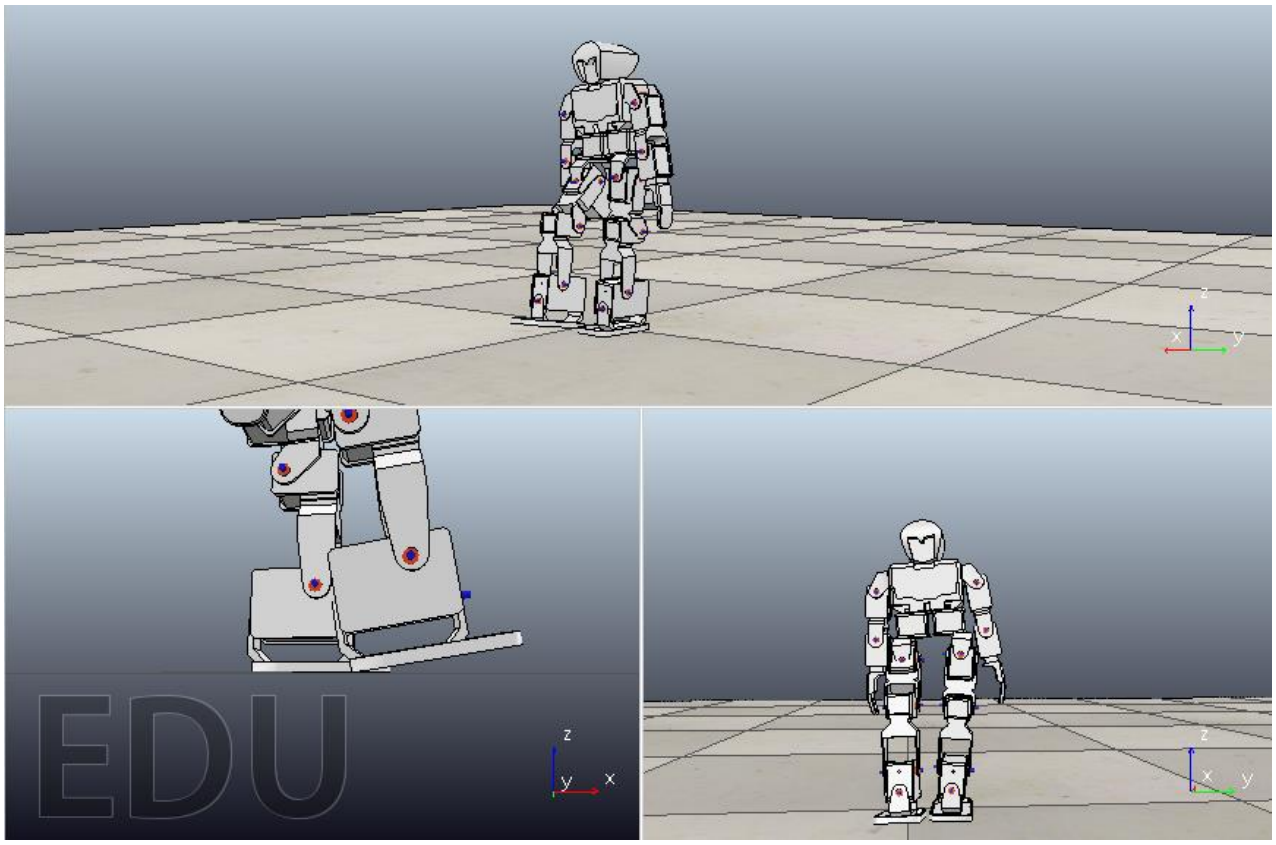


Figure 3-9.Step beginning

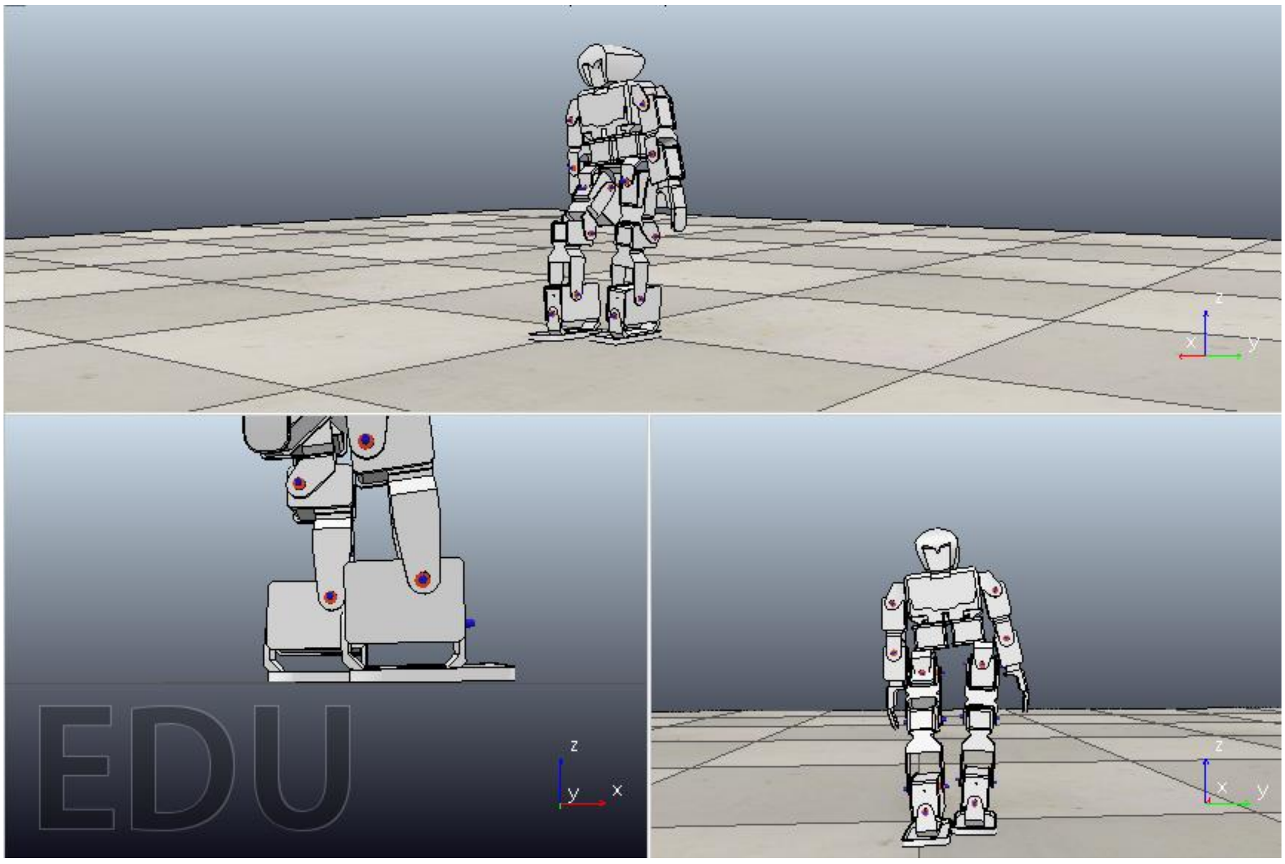


Figure 3-10.Step ending

It is a simple code, but it is tedious and not quite interesting from control's point of view; this is the reason why I'm not going to explain all the steps included in this process. It is implemented in *Biolid_walk_dim.ttt* scene and code can be found in the appendix F clearly commented and detailed.

At this point, I have to find a more effective control strategity. Matlab® seems a good candidate for this task. In the next chapter we are going to see how connection between Matlab® and V-REP® we can establish.

4 API CONECTION WITH MATLAB®

In this chapter, we are going to study th conection beetwen Matlab and V-REP. This conection follows a client-server protocol, where Matlab is the client an V-REP is the server. To understand this protocol in a better way, the user's manual is really helpful, because in it there is a detailed explaination. To establish this connection there are two possibilities:

1. *When V-REP starts **(continuous remote API server service)**:* It is necessary to write a few command lines in the window that appears when software starts and modify certain API libraries. Connection is established In spite of the fact that no simulation is running.
2. *From a script in V-REP **(temporary remote API server service):*** This is the easiest option and the one that has been chosen. In this mode, connection can only be established during simulation. We are going to develop how this conection works.

All the functions which are going to be decribed can be found in the remote API chapter of the user's manual (Bibliography [8]).

4.1 What is it needed?

From Matlab's side we need three files which must ALWAYS be in our workspace:

remoteApiProto.m
remApi.m
remoteApi.dll

These files are libraries of functions we are going to need to send messages from client side (Matlab).
They can be found in the sofware's files in hard disk of our computer.

4.2 How can we send messages?

It will be communicated both sides through port 19999 of our computer. To start or close connection we need the following commands from each side:

V-REP: We need a non-threaned child script like we have done before in which we have to open the connection:

```
if ( simGetScriptExecutionCount ()==0) then
simExtRemoteApiStart (19999); end;
```

This connection will close when simulation stops.

Matlab: To open the connection we need certain API functions which are clearly explained in the user manual:

```
vrep=remApi ( ' remoteApi ' ) ;
vrep.simxFini sh (-1);
clientID=vrep.simxStart ( '127.0.0 .1',19999 , true , true , 5000 , 5 ) ;
```

First command creates API library, second command closes all conections which are opened and we established the connection with the last one. The paremeters are these from left to right:

1. The ip address where the server is located (i.e. V-REP)
2. The port number where to connect
3. If true, then the function blocks until connected (or timed out).
4. If true, then the communication thread will not attempt a second connection if a connection was lost.
5. Timeout in ms:
    a. *if positive:* the connection time-out in milliseconds for the first connection attempt. In that case, the time-out for blocking function calls is 5000 milliseconds.
    b. *if negative:* its positive value is the time-out for blocking function calls. In that case, the connection time-out for the first connection attempt is 5000 milliseconds.
6. Indicates how often data packets are sent back and forth. Reducing this number improves responsiveness, and a default value of 5 is recommended.

At this point, both sides will receive all messages they send between them. If we want to close the connection from Matlab side, we must proceed like this:

```
vrep.simxGetPingTime(clientID ) ;
vrep.simxFinish(clientID ) ;
vrep.delete ( );
```

First command guarantees last message we have sent has had time to arrive. Finally, we close connection. With these steps we have defined all protocol needed in the connection.

5 DYNAMIC MODEL IN MATLAB

At this point we are going to analyze robot dynamic model from Matlab and compare it to V-REP. In first place, we need to identify V-REP joint motor model.

The diagram we are going to follow will be the one that can be found in figure 5-1.

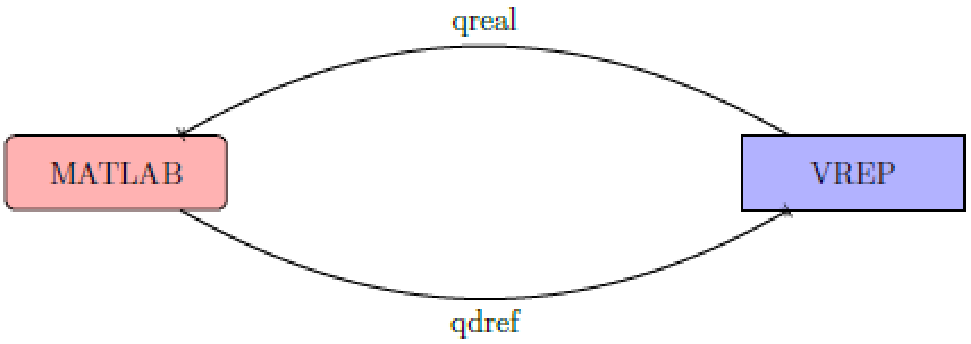


Figure 5-1.Comunication diagram

Matlab function which sends joint velocity is *VREPROBOT.m*. This function receives reference from controller and sends it to server. Besides, this function gains joint position value requesting them to the server. This function and others I am going to describe in this chapter can be found from appendix G to P.

5.1 V-REP joint motor model

In order to identify motor transfer function and parameters, we are going to focus on one joint robot with one link as we can see in figure 5-2.

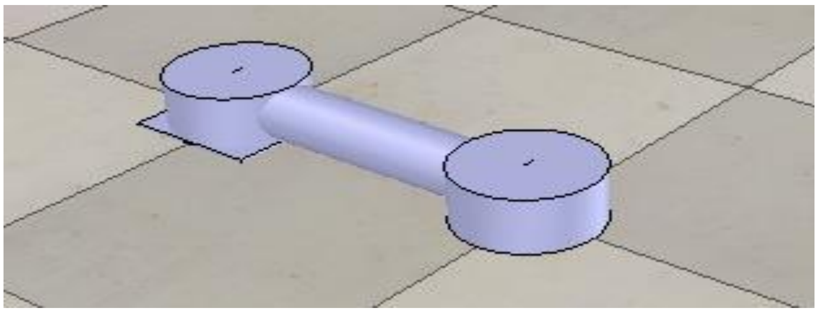


Figure 5-2.One DOF Robot

Firstly, we do not know if inertia of the motor is in relation to the inertia of the link. We need to simulate several experiments with differents inertias of the link and use the results which give a good estimation. We simulate in *open loop* in Matlab and do different simulations changing inertia of the link robot in V-REP, as we can see in figure 5-3.

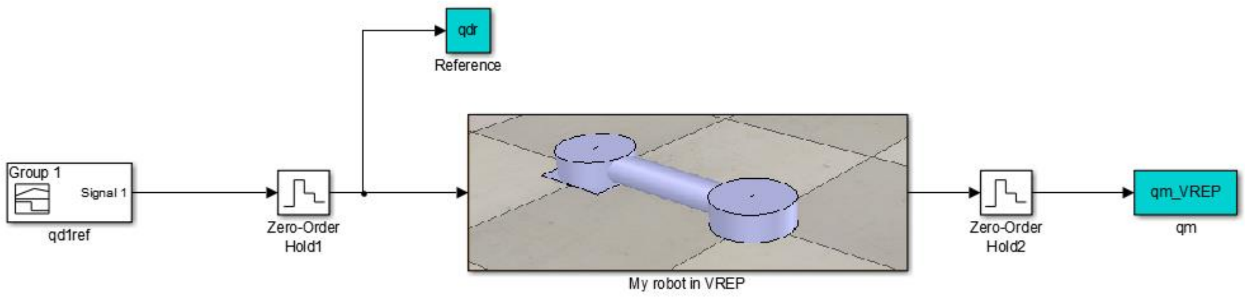


Figure 5-3.Open loop simulations

We need to have setepped expermients; what this means is that it is needed to go from inertias order e-2 to e3 increasing up the order in the next simulation. Summing up, we are to simulate six times, enough time to have good estimations. Now we have joint velocity and joint torque of the motor for inertias orders from e-2 and e0 to the same joint velocity reference:

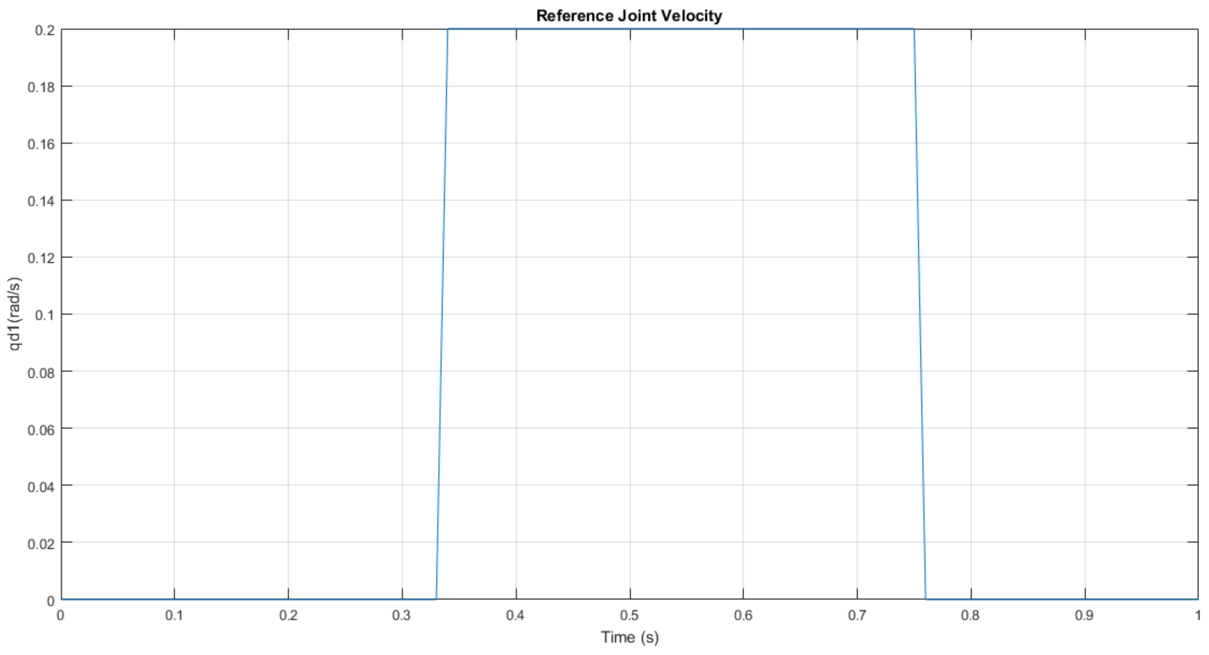


Figure 5-4.Joint Velocity Reference

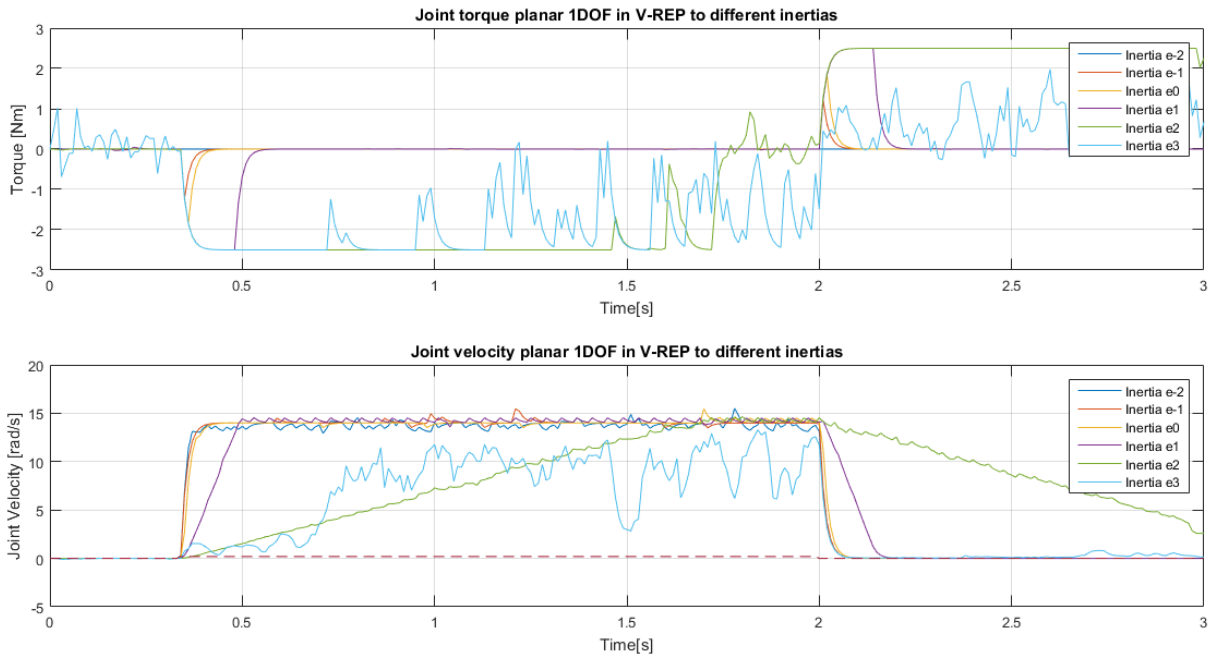


Figure 5-5.Torque and Joint Velocity planar 1DOF robot to differents inertias order

From the results of the simulation it can be concluded that it is a motor controllated by a proportional.To identify all motor parameters and transfer function it is clear that follos equations (1).

$$(Jm + J_{LINK}) \cdot qdd + B \cdot qd = Torque$$

$$G(s) = \frac{1}{(Jm + J_{LINK}) \cdot s + B}$$

$$Torque = -Kp \cdot (qd - qd_{REF}); N = \frac{qd}{qd_{REF}} \qquad (1)$$

$$Torque = Kp \cdot qd \cdot \left(\frac{1}{N} - 1 \right)$$

Where *Jm* is the inertia of the motor [Nm], *Jlink* is the inertia of the link [Nm], *B* is the viscosity of the motor [Nm/(rad/s)], N is relation of transmission and Kp is the proportional action of the motor.

From the results of the simulations changing Jlink for the different experiments, it can be concluded that the parameters of the motors are:

Inertia=0.0765 [Nm]

Viscosity= 6.9577 [Nm/(rad/s)]

Reduction factor=70

$$Kp \cdot \left(\frac{1}{N} - 1 \right) = 1.4e7$$

It is given by a first orden system like this:

$$G(s) = Kp \cdot \left(\frac{1}{N} - 1 \right) \cdot \frac{0.1437}{0.011s + 1} \tag{2}$$

It is clear now that inertia of the link must be higher than the inertia of the motor to simulate dynamic robot model properly and not to simulate motor model. Due to this conlusion, we have to increase our inertia of the link to have a realistic simulation. We are going to change order of inertia to e0.

Now we have to build our dynamic model in Matlab. Peter Corke's robotics toolbox is the easiest way to change dynamic model parameters and simulation. We program our robot with the features we have mentioned as it can be seen in figure 5-6.

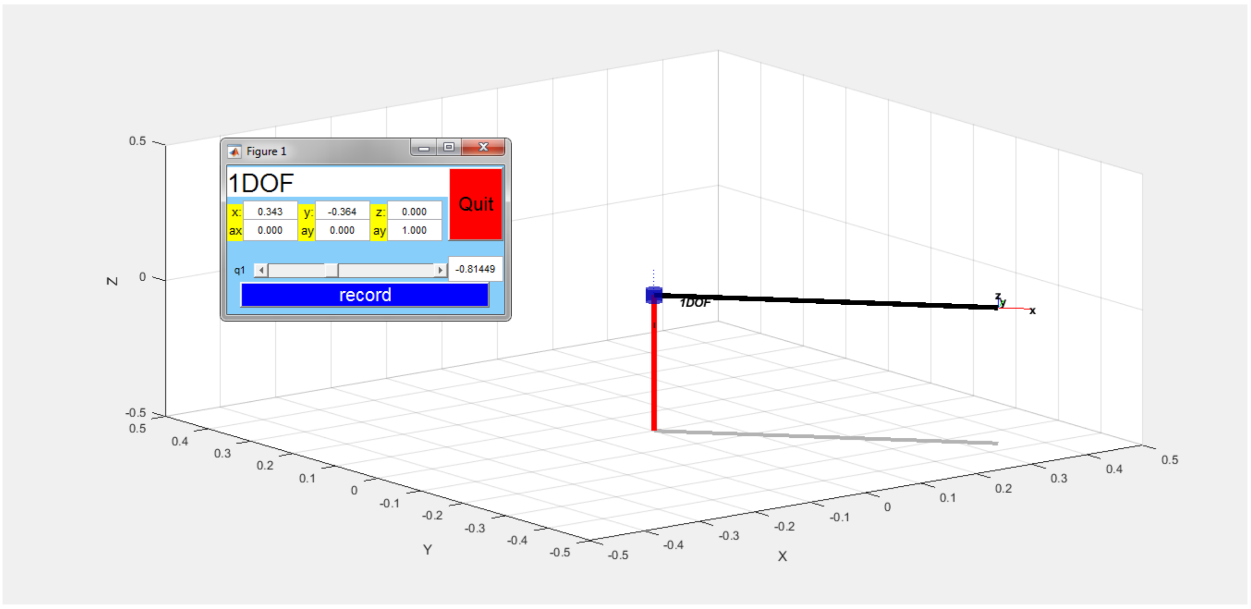


Figure 5-6.Peter Corke one DOF robot

We simulate in the same we did with V-REP robot as we can see in figure 5-7.

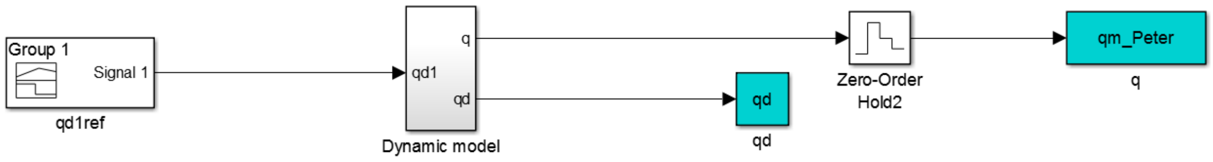
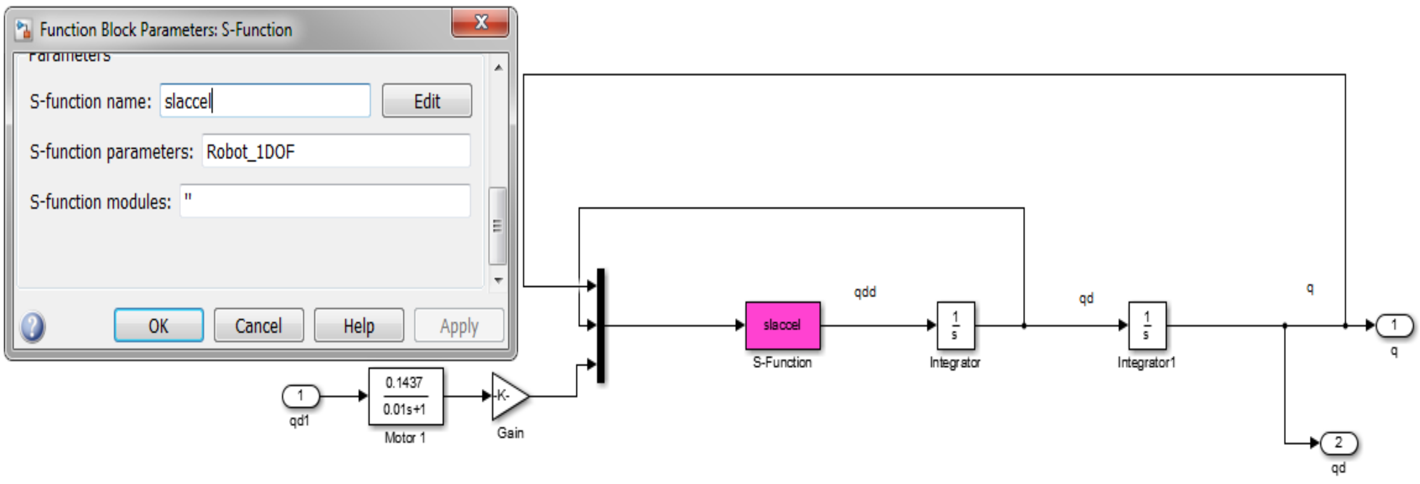


Figure 5-7.Open loop simulations

To calculate robot joint velocity, we are going to use *slaccel* function. This S-function computes robot joint acceleration with joint position, velocity and torque of the motor as inputs as it can be seen in figure 5-8.


Function Block Parameters: S-Function

Parameters

S-function name: slaccel

S-function parameters: Robot_1DOF

S-function modules: ''

Figure 5-8.Dynamic model block

The result for our adjustment can be found in figure 5-9.

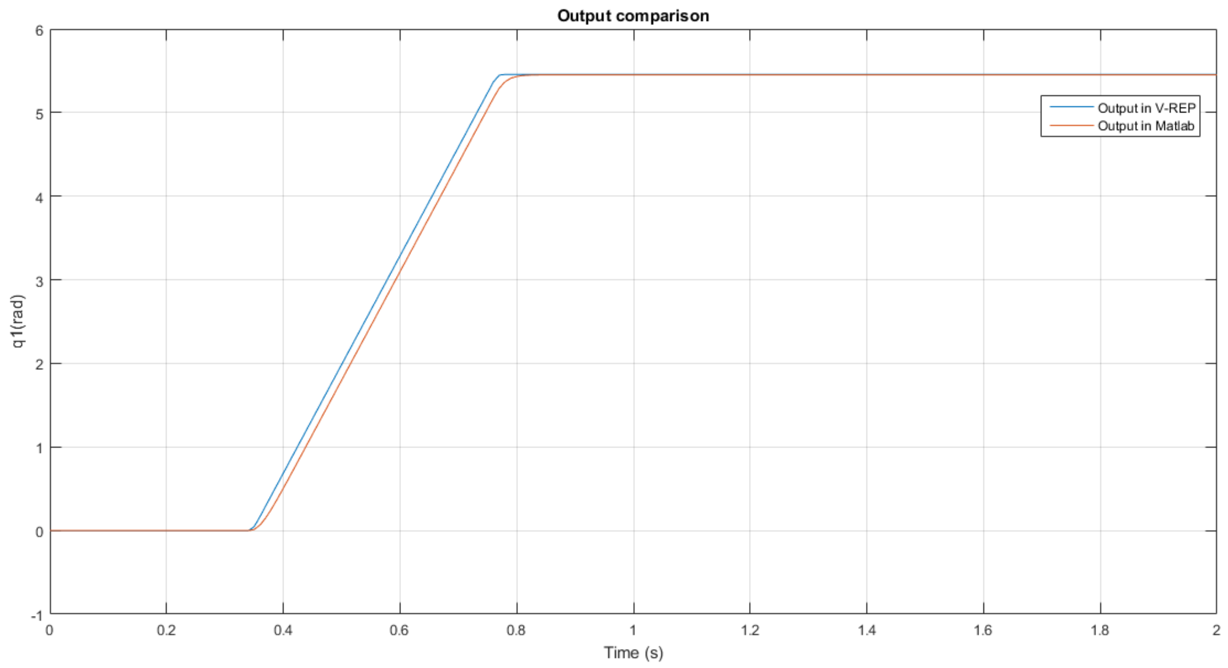


Figure 5-9.Output joint position comparision

5.2 2 DOF dynamic control

We look for control diagram which has been exposed in the second chapter:

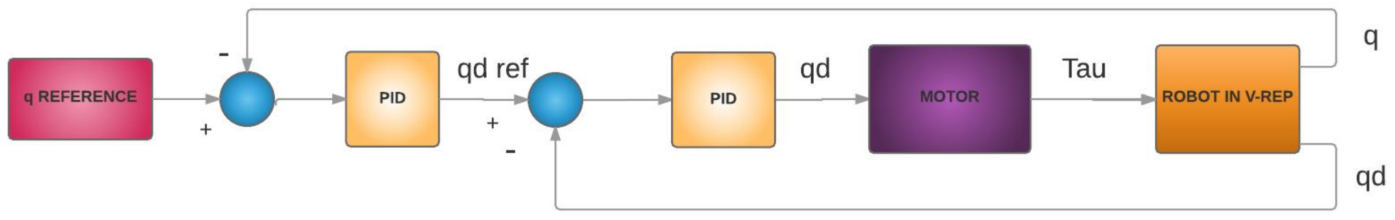


Figure 5-10.Dynamic control diagram

It does not exist any API command to set joint torque directly, you can only get it. Due to this fact, we are going to simulate motor block in V-REP. It is the same for joint velocity, so we will have to derivate position. Our discrete control model in Matlab now can be seen in figure 5-11.

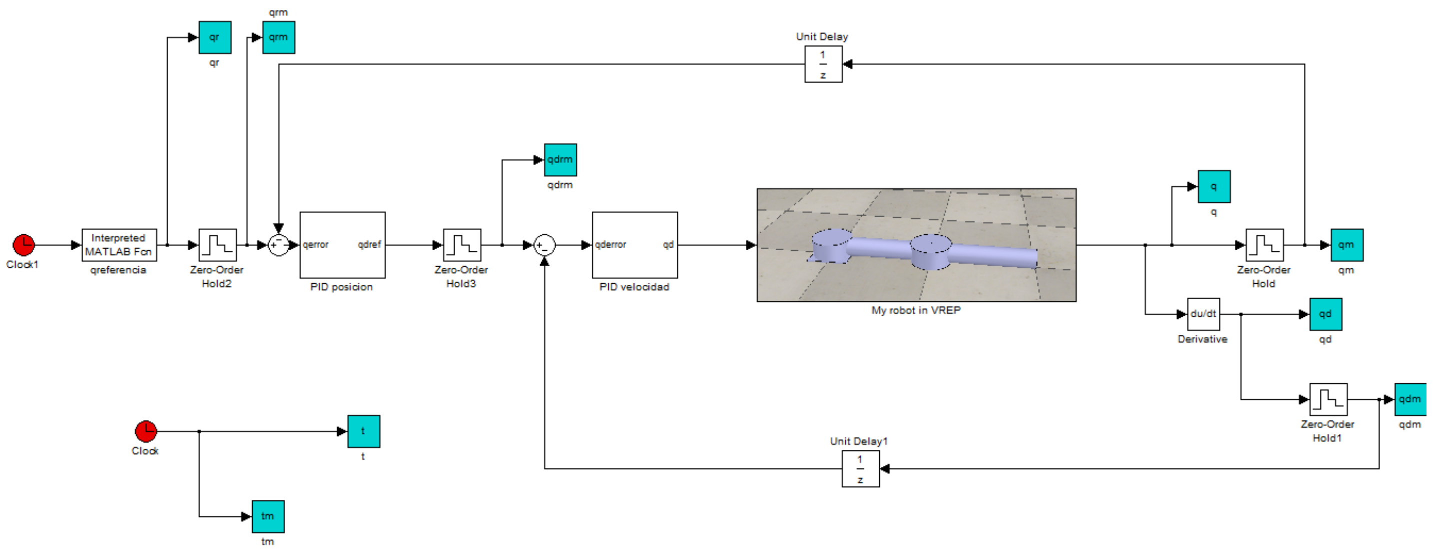
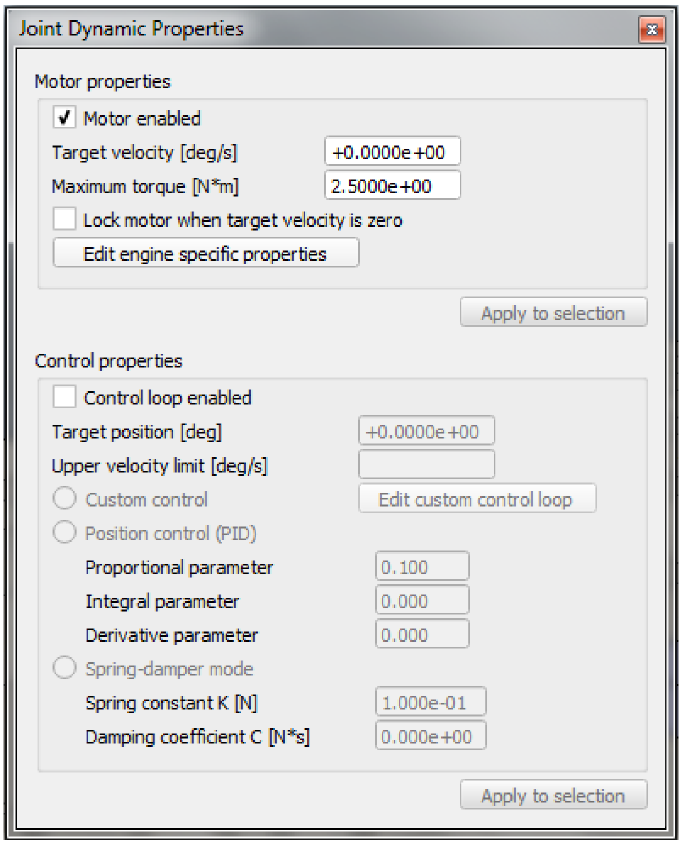


Figure 5-11.Initial dynamic control diagram in Matlab

Nevertheless, to apply this diagram we must disable *Motor enabled* option in joint dynamic properties to velocity controller works as we wish. If this option es enabled, a velocity controller will be activated which receives *TargetVelocity* as reference which will foot the reference we pass in our model.Furthermore, if we disable motor option, we will not be able to apply joint velocity to joint motor with API commands, because it is not the required configuration for the joints. We have to leave velocity control to V-REP, here it is another limitation. Each joint will have the configuration we can found in figure 5-12.


Joint Dynamic Properties

Motor properties

- [✓] Motor enabled
- Target velocity [deg/s]: +0.0000e+00
- Maximum torque [N*m]: 2.5000e+00
- [ ] Lock motor when target velocity is zero
- Edit engine specific properties

Apply to selection

Control properties

- [ ] Control loop enabled
- Target position [deg]: +0.0000e+00
- Upper velocity limit [deg/s]:
- ○ Custom control — Edit custom control loop
- ○ Position control (PID)
  - Proportional parameter: 0.100
  - Integral parameter: 0.000
  - Derivative parameter: 0.000
- ○ Spring-damper mode
  - Spring constant K [N]: 1.000e-01
  - Damping coefficient C [N*s]: 0.000e+00

Apply to selection

Figure 5-12.Joint Dynamic Properties to control in Matlab

Finally, our control in Matlab is the one we can see in figure 5-13.

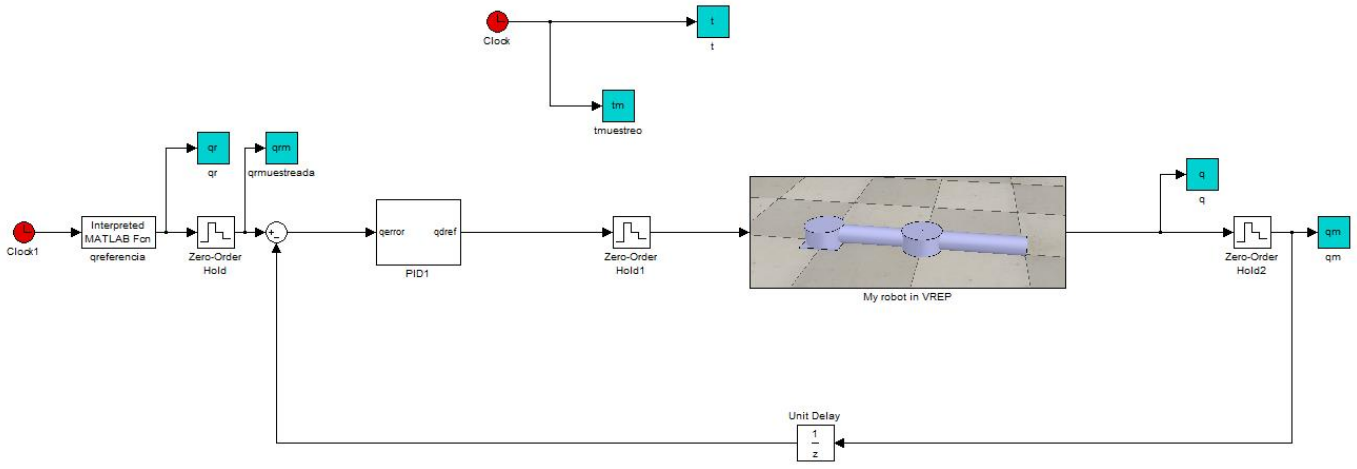


Figure 5-13.Final dynamic control diagram in Matlab

I apply a joint path which will start in 0 [rad] and it will end in 0.8[rad]. My controller will be a proportional controller with proportional constant terrm equal to 0.2[(rad/s)/rad].

It is interesting to say that it is necessary to enable *Real Time Simulation* option in V-REP to Matlab and V-REP to have the same time intervals.

The step that follows, now that we have identified motor model, is to compare the results we will obtain with dynamic control of 2 DOF robot in V-REP to Matlab. First, we are going to build 2 DOF dynamic model in Matlab as we did with 1 DOF. This robot can be seen in figure 5-14.

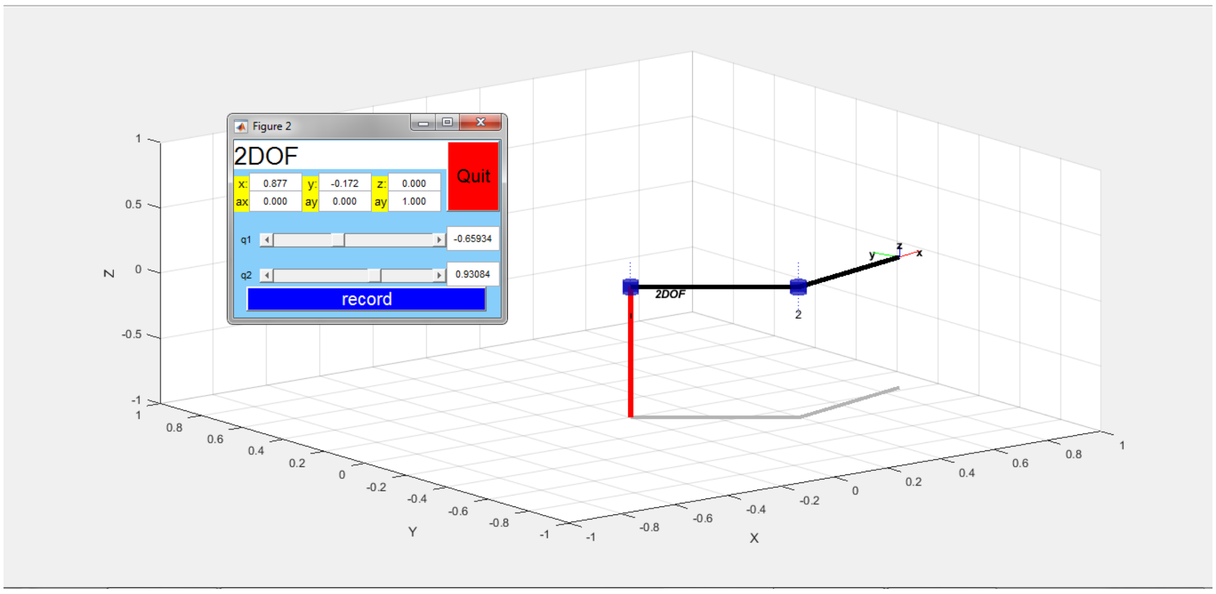


Figure 5-14.Peter Corke 2 DOF robot

And now we simulate them and represent them as it can be seen in figure 5-15.

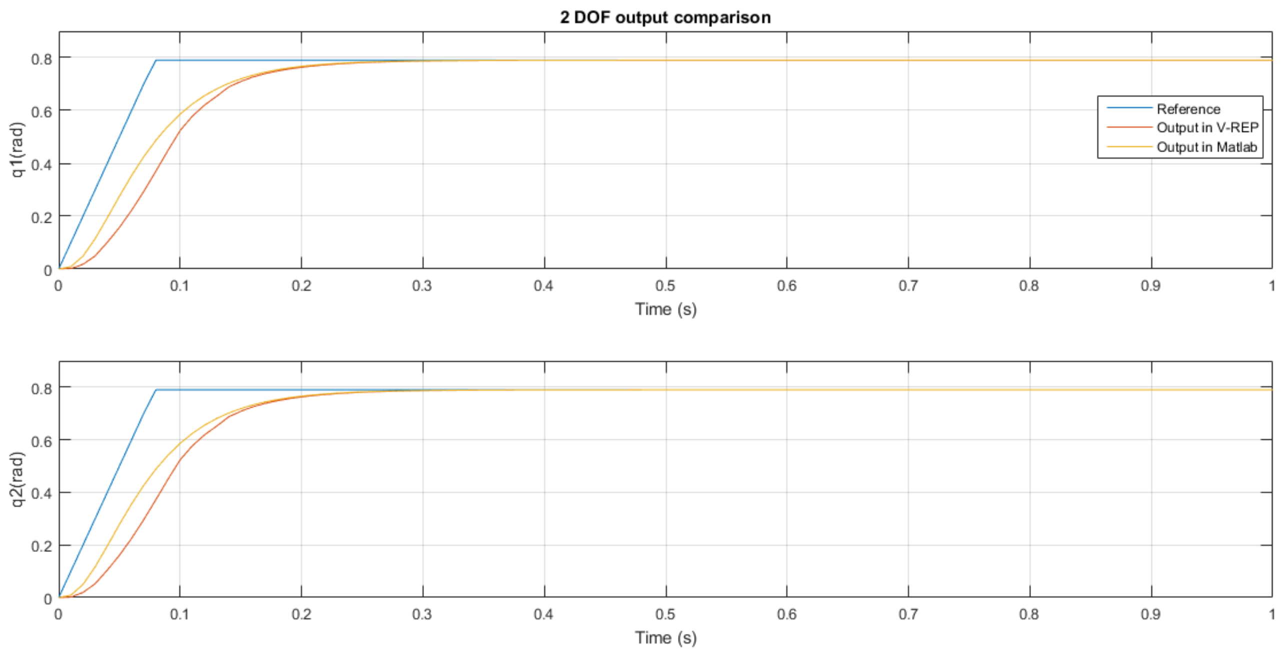


Figure 5-15.Output joint position comparision

In conclusion, it has not been easy to estimate dynamic model, but we have obtained a really good estimation for motor model of V-REP and we have finally understood all V-REP dynamic.

6 KINEMATICS CONTROL FROM MATLAB®

6.1 Pseudo inverse Kinematics control to planar 2 DOF robot

We are going to open software options range and we are going to implement our own kinematics control by Pseudoinverse. Between Matlab and V-REP we want to send and receive the same data, so we will use the function VREPROBOT.m. The control diagram we are going to follow will be the one we can see in figure 6-1.

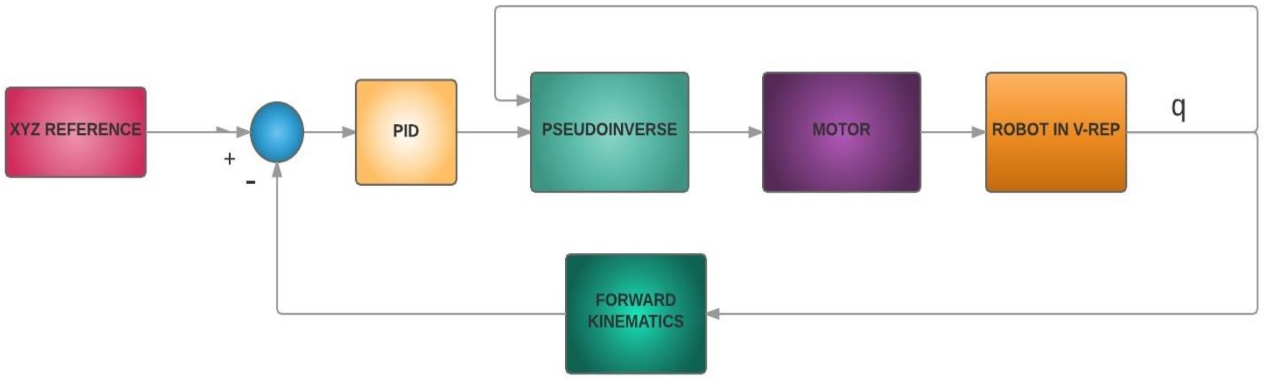


Figure 6-1.Kinematics control diagram-

Pseudoinverse algotithm is given by equations (3)

$$\begin{bmatrix} \mathrm{q1d} \\ \mathrm{q2d} \end{bmatrix} = J^{-1}_{PSEUDO} \cdot \begin{bmatrix} \mathrm{Vx} \\ \mathrm{Vy} \\ \mathrm{Vz} \end{bmatrix} \qquad (3)$$

$$J^{-1}_{PSEUDO} := J^{T} \cdot (J \cdot J^{T})^{-1}$$

It will be needed to calculate the pseudoinverse several times during the simulation, so this task is going to be carried out by the command *pinv* of Matlab, whish is faster.

As it has been said before, we are going to leave V-REP motor block simulation. This diagram implemented in Matlab can be seen in figure 6-2.

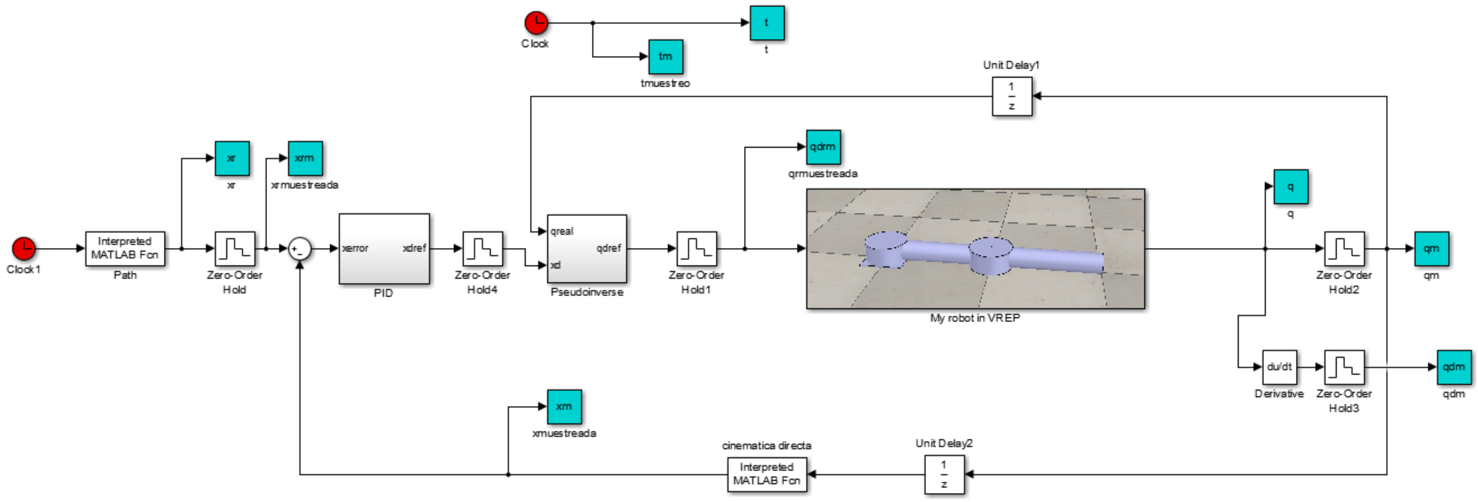


Figure 6-2.Kinematics control diagram in Matlab

To demonstrate its operation, a simulation in which a circle path reference is received will be carried out. Position controller that is going to implementated will be a PI with Ti=100 [s] and Kp=0.5[(m/s)/m]. To explore V-REP more, I am going to add a dummy at the end of the robot and I am going to plot XY postion in two different graphs as we can see in figure 6-3.

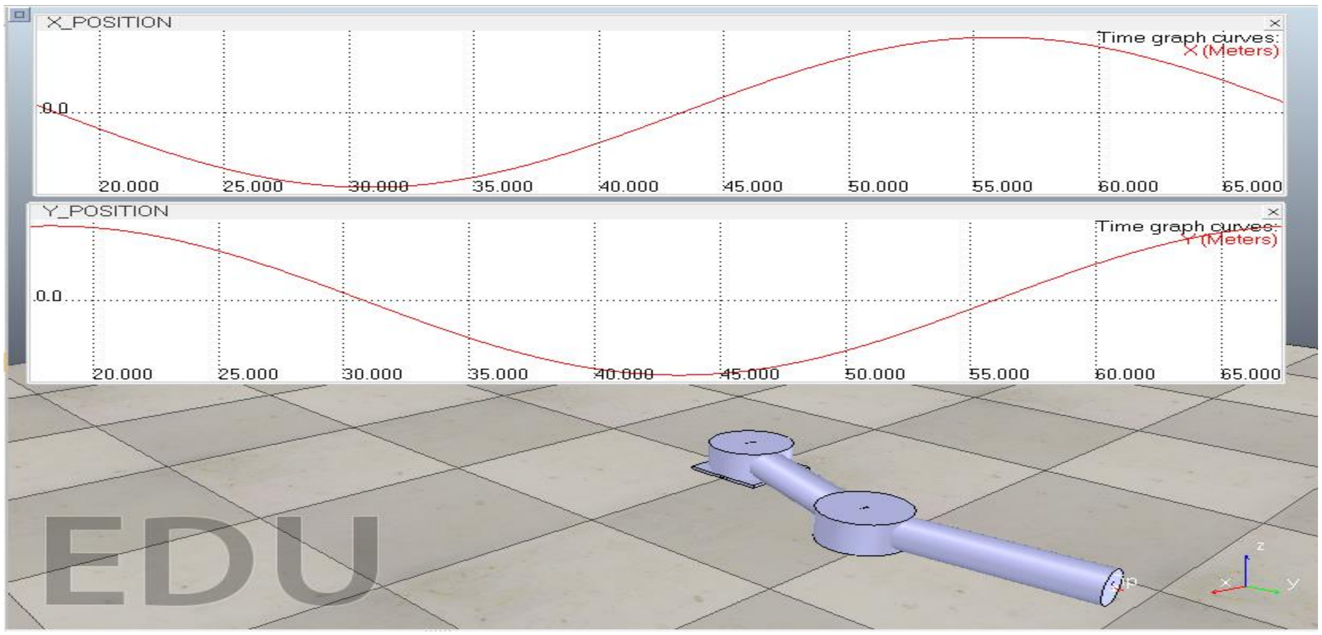


Figure 6-3.Plotting results in V-REP

Finally, the results of the complete simulation can be found in figures 6-4 and 6-5.

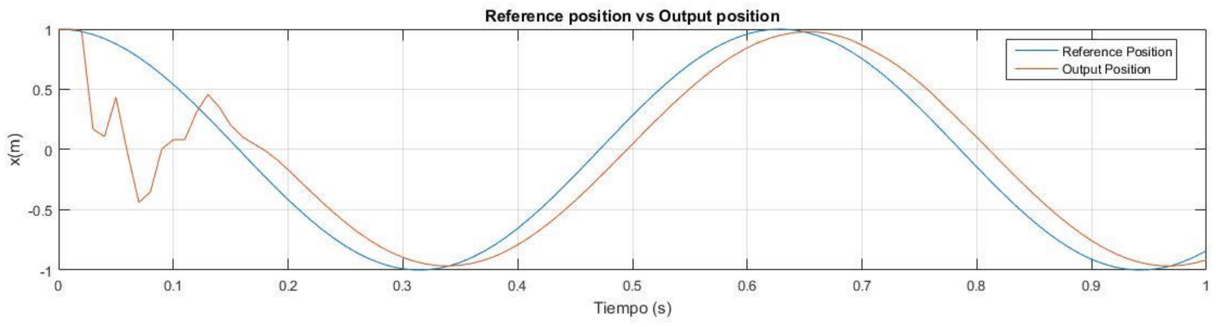


Figure 6-4.Reference versus Output XY position

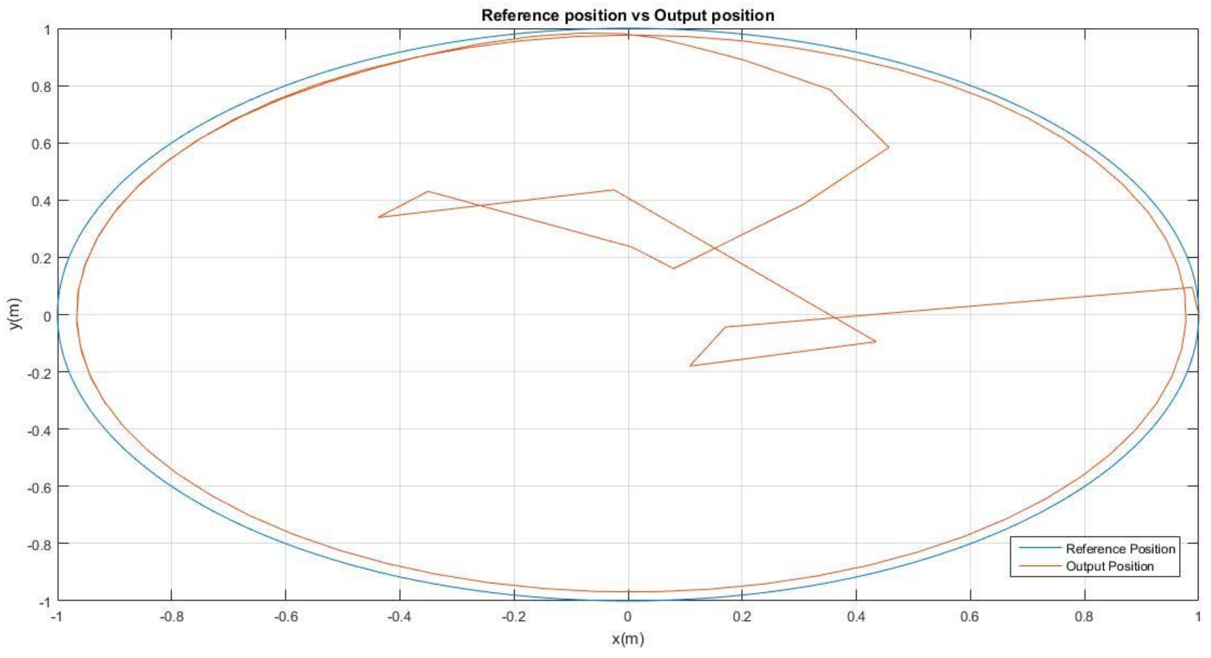


Figure 6-5.Reference versus Output XY position

6.2 DLS kinematics control to 4 DOF robot

To study all options V-REP brings, we are going to implement DLS method to a redundant robot such as 4 DOF robot:

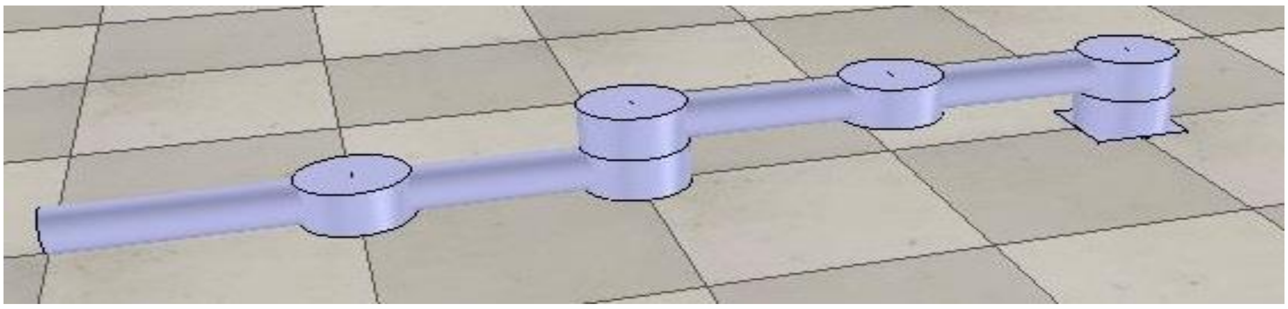


Figure 6-6.4 DOF planar robot in V-REP

With this new configuration, we need to be specially carefull about being close to singular configurations. In order to have acceptable results, we are going to improve the DLS algorithm V-REP brings. The DLS algorithm is given by:

$$J_{DLS}^{-1} := J^T \cdot (J \cdot J^T + k^2 \cdot I)^{-1} \qquad (4)$$

Where *k* is damping factor and *I* is identity matrix. In our case, k will not be a statict value like in V-REP. Damping factor has a maximum value k_0 at singular configurations and zero in other case. So it should be a variable that must be shape for singular configurations and theirs neighbourhood:

$$k := k_0 \cdot \exp\left(-\frac{\det(J \cdot J^T)}{2 \cdot \varepsilon^2}\right) \qquad (5)$$

Where ε is a shaping factor which is between 0 and 1.

We practically have the same diagram in Matlab; the only change being the jacobian pseudoinverse code. Now, we have two new design parameters to obtain better results as we can see in figure 6-7-.

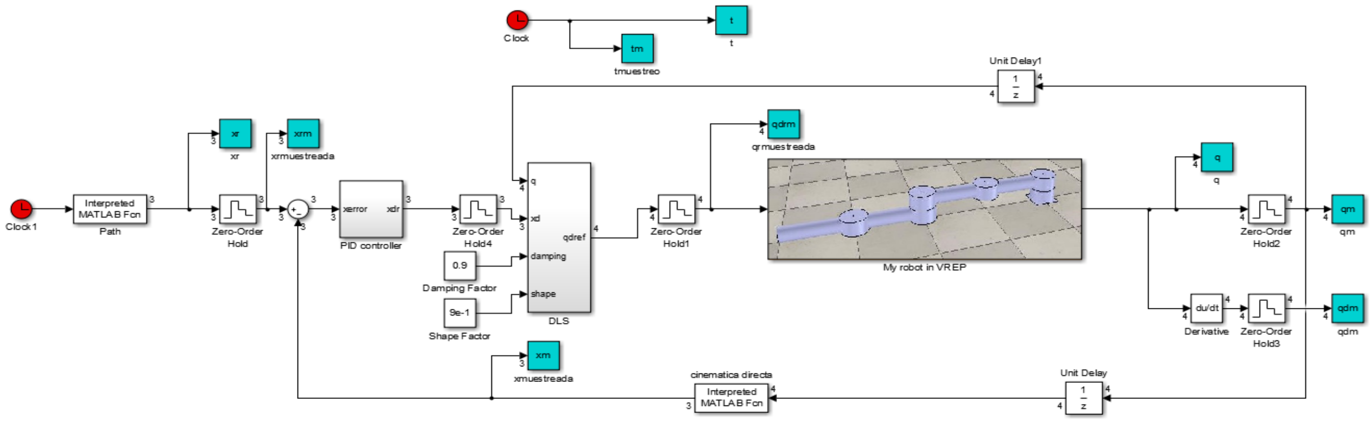


Figure 6-7.DLS kinematics control diagram in Matlab

The position controller that is going to be implemented will be a PI with Ti=100 [s] and Kp=1.2[(m/s)/m]. DLS algorithm will have a damping factor equal to 0.9 and shape factor 0.9. Finally, the results of the complete simulation can be found in figures 6-8 and 6-9.

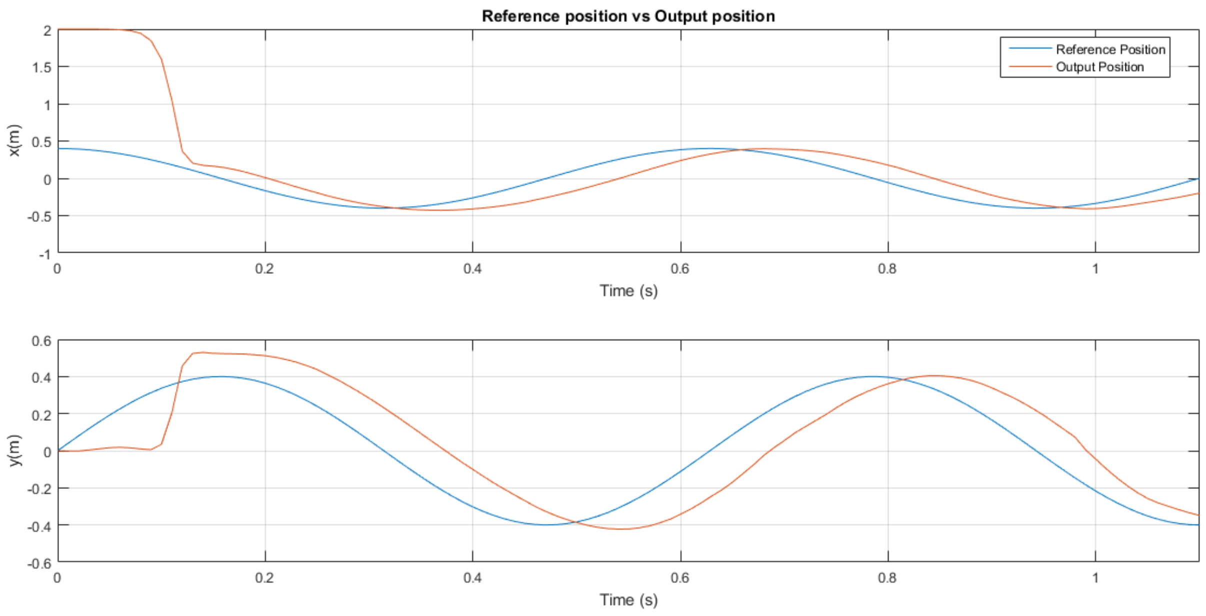


Figure 6-8.Reference versus Output XY position

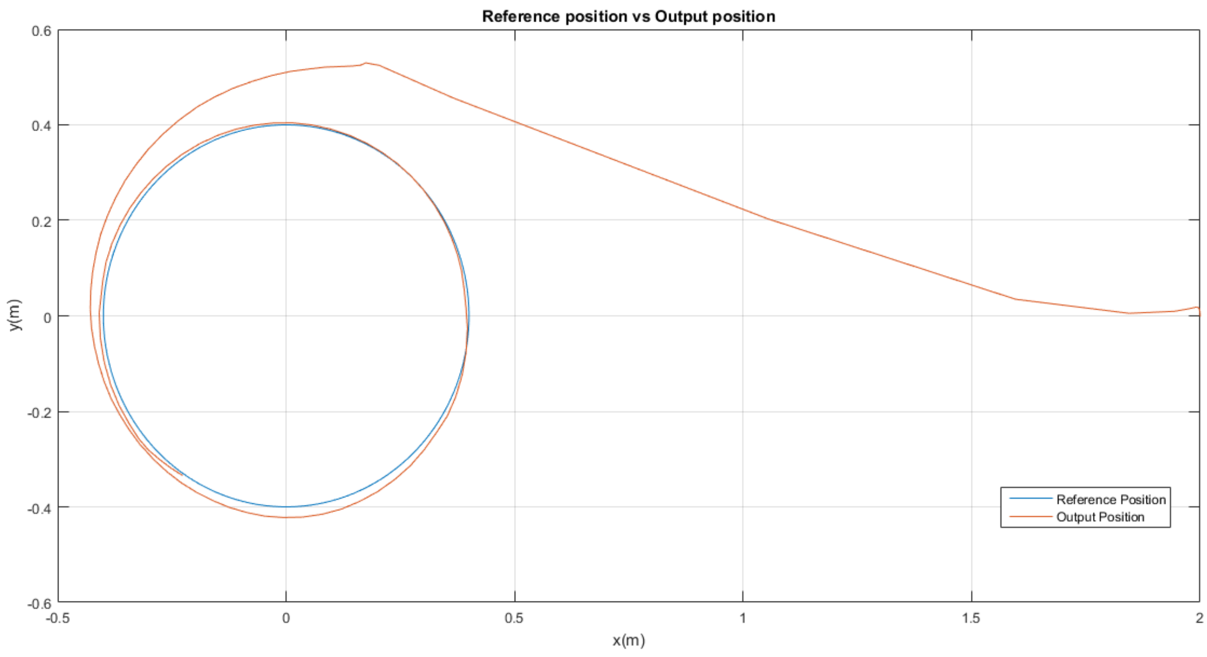


Figure 6-9.Reference versus Output XY position

7 KINECT SENSOR AND V-REP

OUr final experience with V-REP will be to connect XBOX's Kinect sensor to V-REP in order to handle every robot in V-REP with our own body. It is a matter of great interest, due to the fact that the user has de possibility to plug the physical robot in V-REP, and it can be teleoperated with the user's own body.

Figure 7-1.Kinect sensor

To connect Kinect to V-REP, we are going to need two softwares in their first versions, OpenNI and NITE, in adittion to Kinect's drivers. Once all are instaled, we need to copy the following files in our VREP's installation directory in our PC:

Glut32.dll (In its latest version)

OpenNI.dll

SamplesConfig.dll

The problem here is that drivers application V-REP executes *(KinectServer.exe*) are deprecated and they cannot be found easily, due to the fact that software owner website was shut down. In the same way, softwares before mentioned (OpenNI and NITE) must be in their first version. OpenNI2 and NITE2 are **not** valid. I have read through several topics in Coppelia Forum and the main option is to modify the code of the application *KinectServer.exe* which can be found in VREP's installation directory. To check this, questions were asked in the forum, and this was the answer received from Coppelia:

http://www.forum.coppeliarobotics.com/viewtopic.php?f=9&t=6671&p=26600#p26600

Several people have had whis problem, and nobody has been able to solve it yet. Nevertheless, looking for in a deeper way, I have found this link in which you can download all old versions and it is explained why they can be found so easily:

https://fivedots.coe.psu.ac.th/~ad/kinect/installation.html

We are now able to test it in V-REP. We add in our empty scene a new element named *interface to kinect.ttm*. This element has a script which aks for information about our body pose Kinect provides to the server. Our body pose is given a by a simple skeleton. Basically, it is executed *KinectServer.exe* application. All connected follows the next diagram:

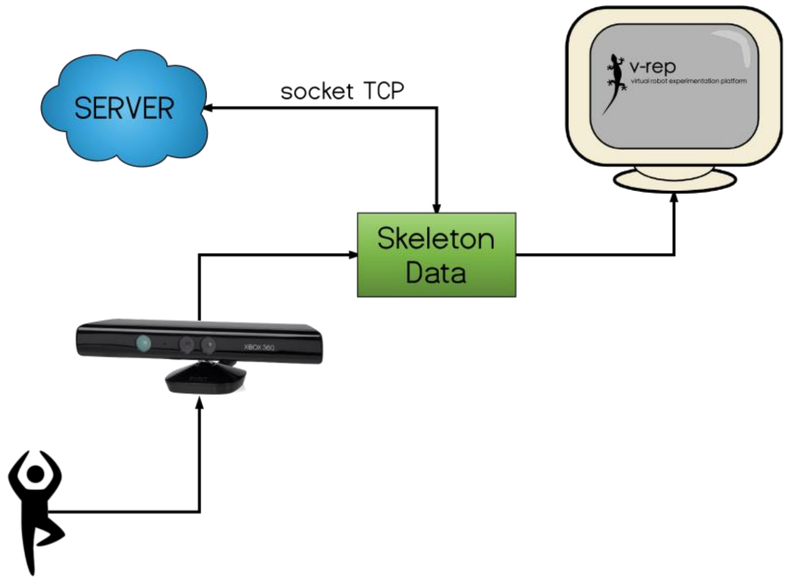


Figure 7-2.Connection diagram

If we run our scene, we will have the result which can be seen in figure 7-3.

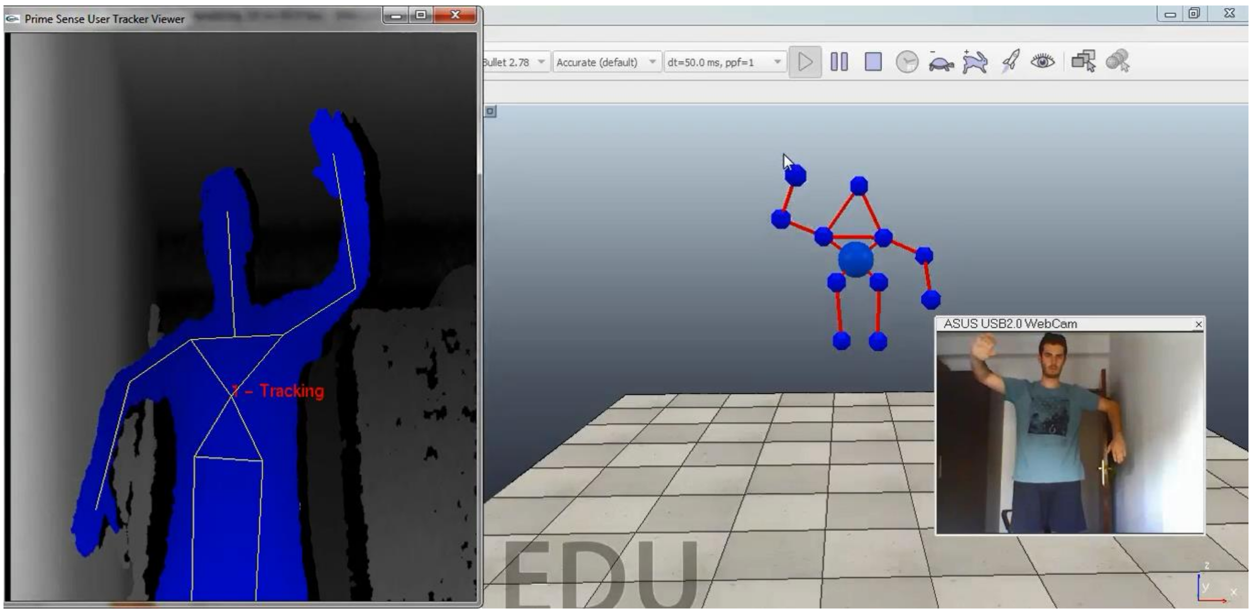


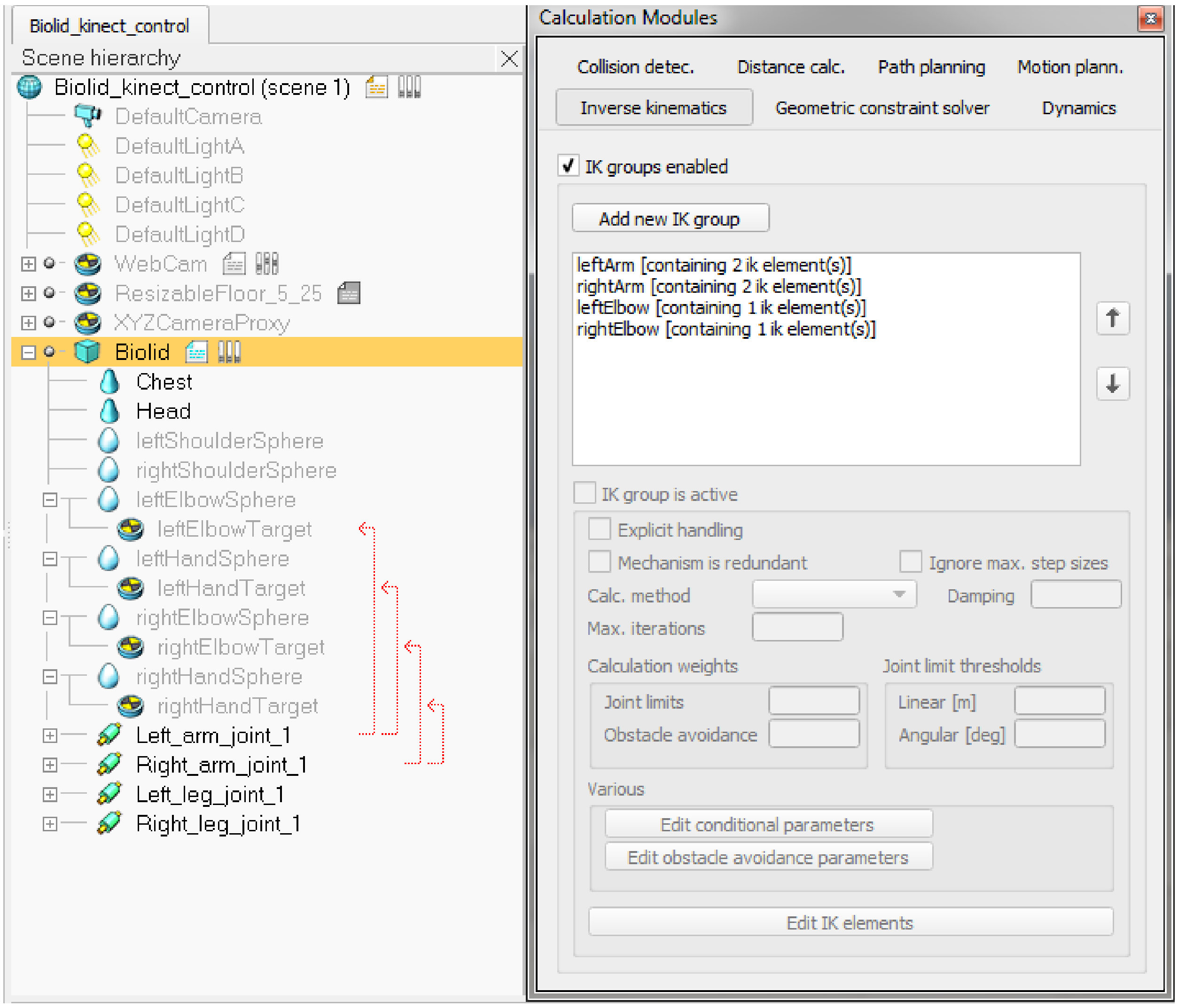
Figure 7-3.Kinect simulation

Which follows is to associate this skeleton to the humanoid robot we have studied.

7.1 Controlling Biolid Humanoid with Kinect Sensor

I am going to associate each point from the skeleton; we have handled before with kinect sensor, with our Biolid Humanoid. In my case, I am only going to associate arms joints. Once they are associated, Biolid will follow the variations in position of the points with an inverse kinematics joint control mixed with torque joint control.

To connect with the *KinectServer.exe* application, I am going to reuse the code we can find in the *interface to kinect.ttm*. To associate each joint with the skeleton, there is a scene, which is proporcionated by V-REP, named *astiKinectControl.ttt*. Nevertheless, this code has several errors. One of them is to define certain variables as local when they are global. An example of this clearly happens in lines 233 and 234 of the code. Once we have corrected the code, we need to define several *IK groups* which contain the dummies related to the skeleton:



leftArm [containing 2 ik element(s)]
rightArm [containing 2 ik element(s)]
leftElbow [containing 1 ik element(s)]
rightElbow [containing 1 ik element(s)]

Figure 7-4.Hierarchy three and *IK groups* in our scene

We are going to set DLS as calculation method, 6 as maximum number of iterations and 0.1 of damping factor for each *IK group*.

Finally, we obtain the behavior we were looking for as we can see in figure 7-5:

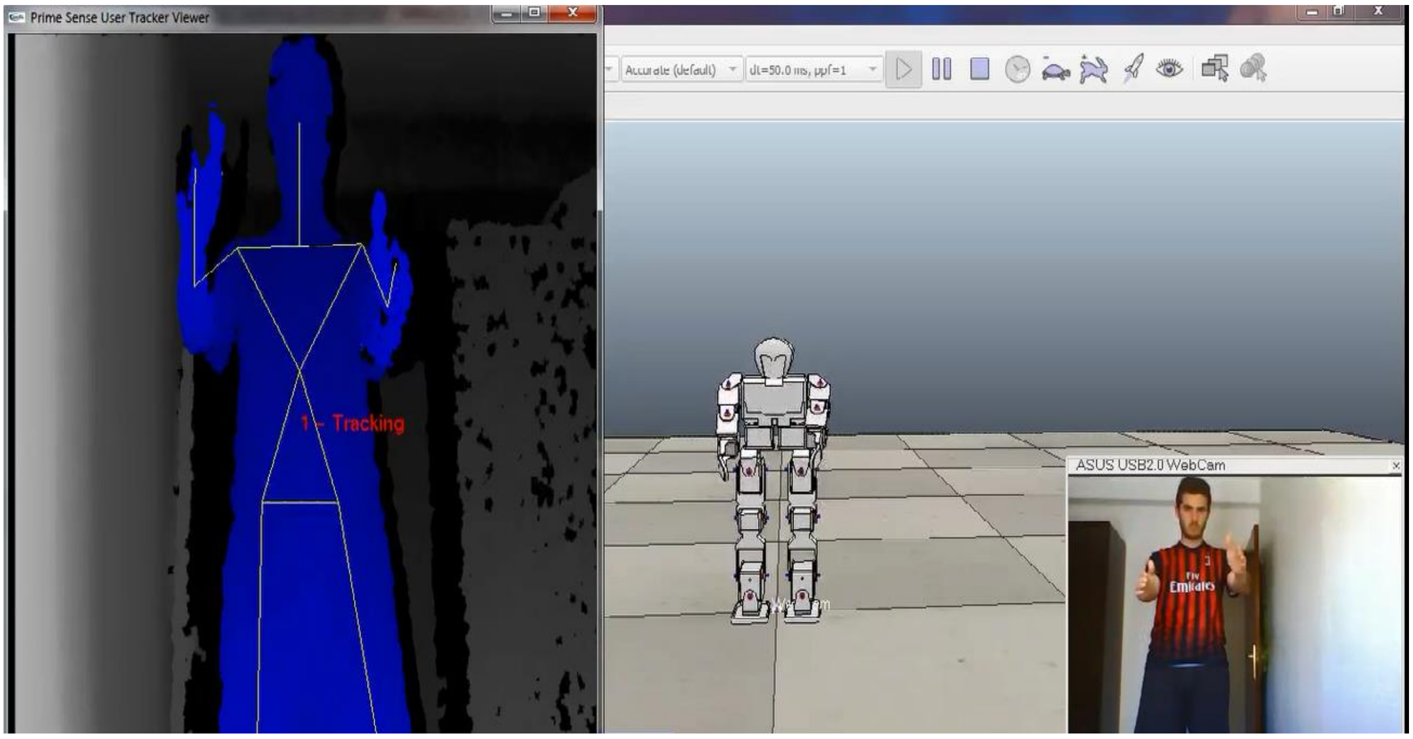


Figure 7-5.Kinect simulation

This scene correctly implemented can be found in *Biolid_Kinect_Control.ttt* and the code of this scene in the appendix Q.

8 FINAL CONSIDERATION AND FUTURE WORKS

The main goals achieved with the present investigation was to study V-REP in a deep way and communicate it with Matlab, new robot configurations and new robot control techniques for me.

In order to make control humanoid robot walks, one of the future works to do will be to develop a kinematics control as we have seen in this thesis. In addition, it can be explored connections API connections with others softwares such as ROS. Furthermore, it will be interesting to explore all V-REP such as object detection or path planning.

Finally, it can be also challenging to connect physical humanoid robot to V-REP and teleoperated it with Kinect sensor.

BIBLIOGRAPHY

[1] Siciliano, B.; Sciavicco, L.; Villani, L. & Oriolo, G. *Robotics. Modelling, Planning and Control*. Springer, 2009

[2] Serena Ivaldi, V. P. & Norix, F. *Tools for dynamics simulation of robots: a survey based on user feedback*. arXiv:1402.7050 [cs.RO], 2014

[3] Robotis.CO. L. *Biolid Premium Kit Manual web site*.
Link: https://issuu.com/ro-botica/docs/quick_start_v1_low_es

[4] Coppelia Robotics. *V-REP 3.4.0 download web site*.
Link: http://www.coppeliarobotics.com/downloads.html

[5] Coppelia Robotics. *Forum web site*.
Link: http://www.forum.coppeliarobotics.com/

[6] Coppelia Robotics. *API functions web site.*
Link: http://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm

[7] Coppelia Robotics. *Remote API functions to Matlab web site.*
Link: http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm

[8] Coppelia Robotics. *V-REP user guide web site.*
Link: http://www.coppeliarobotics.com/helpFiles/

[9] Vázquez Martínez, R. *Trabajo de Fin de Grado en Ingeniería Aeroespacial Intensificación de Vehículos Aeroespaciales. Diseño de un sistema de vuelo para un robot humanoide*. Universidad de Sevilla, 2014

[10] M.I. Sánchez, J.Á. Acosta and A.Ollero, "Integral Action in First-Order Closed-Loop Inverse Kinematics. Application to Aerial Manipulators", in *IEEE International Conference on Robotics and Automation*. Seatle, Washington, May 26-30, 2015.

[11] Davison, A. *OpenNI/NITE Installation on Windows*.
Link: https://fivedots.coe.psu.ac.th/~ad/kinect/installation.html

[12] Corke, Peter, *Robotics, Vision and Control*, Springer, 2011

APPENDIX

Appendix A: Dynamic control with slider for 2 DOF robot in V-REP:

```
if (sim_call_type==sim_childscriptcall_initialization) then
    --Handle user interface:
    ui=simGetUIHandle("Control_articulaciones")

    --Set joints ranges:
    minVal={0,            -- q1
            0}            -- q2

    rangeVal={  math.pi/4,    -- q1
                math.pi/4}    -- q2
    --Sliders positions in the UI:
    uiSliderIDs={3, 4}

    --Handle the joints:
articulaciones={simGetObjectHandle("Joint_1"),simGetObjectHandle("Joint_2")}

    --Apply ranges to UI:
    simSetUISlider(ui,uiSliderIDs[1],(simGetJointPosition(articulaciones[1])-
minVal[1])*1000/rangeVal[1])
    simSetUISlider(ui,uiSliderIDs[2],(simGetJointPosition(articulaciones[2])-
minVal[2])*1000/rangeVal[2])

end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then

    --Set joint values:
simSetJointTargetPosition(articulaciones[1],minVal[1]+simGetUISlider(ui,uiSli
derIDs[1])*rangeVal[1]/1000)

simSetJointTargetPosition(articulaciones[2],minVal[2]+simGetUISlider(ui,uiSli
derIDs[2])*rangeVal[2]/1000)

end
```

Appendix B: Obtain Jacobian for 2 DOF robot in V-REP:

```
if (sim_call_type==sim_childscriptcall_initialization) then
--Get Inverse Kinematics Group:
    ik=simGetIkGroupHandle("IK_2GOF")

end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then

--OBTAIN Jacobian:
--Handle ik group:
           simHandleIkGroup(ik)
            local jaco =simComputeJacobian(ik,0, NULL)
            local jacobiano,indice=simGetIkGroupMatrix(ik,0)

--index[1] represents the number of jacobian's rows.
--index[2] represents the number of jacobian's columns.

-- Jacobian data is organized like that:
--
[row1,column1],[row2,column1],..,[rowN,column1],[row1,column2],[row2,column2]
,...

--Turn the result into a character string to display on command window:
            for i=1,indice[1],1 do
                str=''
                for j=1,indice[2],1 do
                    if #str~=0 then
                        str=str..', '
                    end
                    str=str..string.format("%.1e",jacobiano[(j-
1)*indice[1]+i])
                end
                simAddStatusbarMessage(str)
            end
    end
```

Appendix C: Arms dynamic controller with sliders for humanoid robot in V-REP.

```
if (sim_call_type==sim_childscriptcall_initialization) then
    biolid=simGetObjectHandle("Biolid")
    ui=simGetUIHandle("Biolid_arms_control")
    simSetUIButtonLabel(ui,0,simGetObjectName(biolid).." Arms Control")

    minVal={
            0,    -- Left shoulder 1
            0,    -- Left shoulder 2
            0,    -- Left shoulder 3
            0,    -- Right shoulder 1
            0,    -- Right shoulder 2
            0}    -- Right shoulder 3
    rangeVal={
            2*math.pi,    -- Left shoulder 1
            2*math.pi,    -- Left shoulder 2
            2*math.pi,    -- Left shoulder 3
            2*math.pi,    -- Right shoulder 1
            2*math.pi,    -- Right shoulder 2
            2*math.pi}    -- Right shoulder 3
    uiSliderIDs={4,5,6,7,8,9}


rightArmJoints={simGetObjectHandle("Right_arm_joint_1"),simGetObjectHandle("R
ight_arm_joint_2"),simGetObjectHandle("Right_arm_joint_3")}

leftArmJoints={simGetObjectHandle("Left_arm_joint_1"),simGetObjectHandle("Lef
t_arm_joint_2"),simGetObjectHandle("Left_arm_joint_3")}

    --Assign values to the sliders:
    simSetUISlider(ui,uiSliderIDs[1],(simGetJointPosition(rightArmJoints[1])-
minVal[1])*1000/rangeVal[1])
    simSetUISlider(ui,uiSliderIDs[2],(simGetJointPosition(rightArmJoints[2])-
minVal[2])*1000/rangeVal[2])
    simSetUISlider(ui,uiSliderIDs[3],(simGetJointPosition(rightArmJoints[3])-
minVal[3])*1000/rangeVal[3])
    simSetUISlider(ui,uiSliderIDs[4],(simGetJointPosition(leftArmJoints[1])-
minVal[4])*1000/rangeVal[4])
    simSetUISlider(ui,uiSliderIDs[5],(simGetJointPosition(leftArmJoints[2])-
minVal[5])*1000/rangeVal[5])
    simSetUISlider(ui,uiSliderIDs[6],(simGetJointPosition(leftArmJoints[3])-
minVal[6])*1000/rangeVal[6])

end

if (sim_call_type==sim_childscriptcall_cleanup) then
end

if (sim_call_type==sim_childscriptcall_actuation) then

    -- Read desired values from the user control:
simSetJointTargetPosition(rightArmJoints[1],minVal[1]+simGetUISlider(ui,uiSli
derIDs[1])*rangeVal[1]/1000)

simSetJointTargetPosition(rightArmJoints[2],minVal[2]+simGetUISlider(ui,uiSli
derIDs[2])*rangeVal[2]/1000)
```

```
simSetJointTargetPosition(rightArmJoints[3],minVal[3]+simGetUISlider(ui,uiSli
derIDs[3])*rangeVal[3]/1000)

simSetJointTargetPosition(leftArmJoints[1],minVal[4]+simGetUISlider(ui,uiSlid
erIDs[4])*rangeVal[4]/1000)

simSetJointTargetPosition(leftArmJoints[2],minVal[5]+simGetUISlider(ui,uiSlid
erIDs[5])*rangeVal[5]/1000)

simSetJointTargetPosition(leftArmJoints[3],minVal[6]+simGetUISlider(ui,uiSlid
erIDs[6])*rangeVal[6]/1000)

end
```

Appendix D: Right Leg dynamic controller with sliders for humanoid robot in V-REP.

```
if (sim_call_type==sim_childscriptcall_initialization) then
    biolid=simGetObjectHandle("Biolid")
    ui=simGetUIHandle("Biolid_Leg_control")
    simSetUIButtonLabel(ui,0,simGetObjectName(biolid).." Leg Control")

    minVal={
            0,    -- 1
            0,    -- 2
            0,    -- 3
            0,    -- 4
            0,    -- 5
            0}    -- 6
    rangeVal={
            2*math.pi,    -- 1
            2*math.pi,    -- 2
            2*math.pi,    -- 3
            2*math.pi,    -- 4
            2*math.pi,    -- 5
            2*math.pi}    -- 6
    uiSliderIDs={4,5,6,7,8,9}


rightLegJoints={simGetObjectHandle("Right_leg_joint_1"),simGetObjectHandle("R
ight_leg_joint_2"),simGetObjectHandle("Right_leg_joint_3"),
simGetObjectHandle("Right_leg_joint_4"),
simGetObjectHandle("Right_leg_joint_5"),
simGetObjectHandle("Right_leg_joint_6")}

    --Assign values to the sliders:
    simSetUISlider(ui,uiSliderIDs[1],(simGetJointPosition(rightLegJoints[1])-
minVal[1])*1000/rangeVal[1])
    simSetUISlider(ui,uiSliderIDs[2],(simGetJointPosition(rightLegJoints[2])-
minVal[2])*1000/rangeVal[2])
    simSetUISlider(ui,uiSliderIDs[3],(simGetJointPosition(rightLegJoints[3])-
minVal[3])*1000/rangeVal[3])
    simSetUISlider(ui,uiSliderIDs[4],(simGetJointPosition(rightLegJoints[4])-
minVal[1])*1000/rangeVal[1])
    simSetUISlider(ui,uiSliderIDs[5],(simGetJointPosition(rightLegJoints[5])-
minVal[2])*1000/rangeVal[2])
    simSetUISlider(ui,uiSliderIDs[6],(simGetJointPosition(rightLegJoints[6])-
minVal[3])*1000/rangeVal[3])

end

if (sim_call_type==sim_childscriptcall_cleanup) then
end

if (sim_call_type==sim_childscriptcall_actuation) then

    -- Read desired values from the user control:

simSetJointTargetPosition(rightLegJoints[1],minVal[1]+simGetUISlider(ui,uiSli
derIDs[1])*rangeVal[1]/1000)

simSetJointTargetPosition(rightLegJoints[2],minVal[2]+simGetUISlider(ui,uiSli
derIDs[2])*rangeVal[2]/1000)
```

```
simSetJointTargetPosition(rightLegJoints[3],minVal[3]+simGetUISlider(ui,uiSli
derIDs[3])*rangeVal[3]/1000)

simSetJointTargetPosition(rightLegJoints[4],minVal[4]+simGetUISlider(ui,uiSli
derIDs[4])*rangeVal[4]/1000)

simSetJointTargetPosition(rightLegJoints[5],minVal[5]+simGetUISlider(ui,uiSli
derIDs[5])*rangeVal[5]/1000)

simSetJointTargetPosition(rightLegJoints[6],minVal[6]+simGetUISlider(ui,uiSli
derIDs[6])*rangeVal[6]/1000)

end
```

Appendix E: Walking by IK calculation with dummies for humanoid robot in V-REP.

```
if (sim_call_type==sim_childscriptcall_initialization) then
    biolid=simGetObjectHandle("Biolid")
    lFoot=simGetObjectHandle("Left_foot_target")
    rFoot=simGetObjectHandle("Right_foot_target")
    lPath=simGetObjectHandle("Left_foot_path")
    rPath=simGetObjectHandle("Right_foot_path")
    lPathLength=simGetPathLength(lPath)
    rPathLength=simGetPathLength(rPath)
    dist=0

    nominalVelocity=3e-3


end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then

-- Get the desired position and orientation of each foot from the paths:
--Depends on the selected path lenght calculation method.
    t=simGetSimulationTimeStep()*nominalVelocity
    dist=dist+t
    lPos=simGetPositionOnPath(lPath,dist/lPathLength)--return values
position: table of 3 values (x, y and z)
    lOr=simGetOrientationOnPath(lPath,dist/lPathLength)--return values
eulerAngles: table of 3 values (alpha, beta and gamma)

    rPos=simGetPositionOnPath(rPath,dist/rPathLength)
    rOr=simGetOrientationOnPath(rPath,dist/rPathLength)


-- Now transform the absolute position/orientation to position/orientation
relative to biolid
    biolidM=simGetObjectMatrix(biolid,-1)--Specify -1 to retrieve the
absolute transformation matrix
--return value table of 12 numbers:
    --The x-axis of the orientation component is
(matrix[1],matrix[5],matrix[9])
    --The y-axis of the orientation component is
(matrix[2],matrix[6],matrix[10])
    --The z-axis of the orientation component is
(matrix[3],matrix[7],matrix[11])
    --The translation component is
(matrix[4],matrix[8],matrix[12])
    biolidMInverse=simGetInvertedMatrix(biolidM)

    m=simMultiplyMatrices(biolidMInverse, simBuildMatrix(lPos,lOr))
--simMultiplyMatrices: return value the output matrix (the result of the
multiplication: matrixIn1*matrixIn2), same matrix as above.
--simBuildMatrix: return value the transformation matrix
    pos_or_ob=simMultiplyMatrices(biolidM,m)
    lPos={pos_or_ob[4],pos_or_ob[8],pos_or_ob[12]}
    lOr=simGetEulerAnglesFromMatrix(pos_or_ob) --table to 3 numbers
representing the Euler angles
```

```
    m=simMultiplyMatrices(biolidMInverse, simBuildMatrix(rPos,rOr))
    pos_or_ob=simMultiplyMatrices(biolidM,m)
    rPos={pos_or_ob[4], pos_or_ob[8], pos_or_ob[12]}
    rOr=simGetEulerAnglesFromMatrix(pos_or_ob)


-- Apply the desired ABSOLUTE positions/orientations to each foot (to two
dummy objects that are then handled by the IK module)
-- to automatically calculate all leg joint desired values.
    simSetObjectPosition(lFoot, -1,lPos) --Specify -1 to set the absolute
position
    simSetObjectOrientation(lFoot,-1,lOr)

    simSetObjectPosition(rFoot,-1,rPos)
    simSetObjectOrientation(rFoot,-1,rOr)

end
```

Appendix F: Walking by state machine for humanoid robot in V-REP.

```
if (sim_call_type==sim_childscriptcall_initialization) then
    dist=0
    nominalVelocity=3e-2
    case=1
end

if (sim_call_type==sim_childscriptcall_cleanup) then

end

if (sim_call_type==sim_childscriptcall_actuation) then
v=simGetSimulationTimeStep()*nominalVelocity
dist=dist+v

if(case==1)then
if(simGetJointTargetPosition(simGetObjectHandle("Left_leg_joint_6"))<0.09)
then
simSetJointTargetPosition(simGetObjectHandle("Right_leg_joint_6"),
math.pi/72+dist)
simSetJointTargetPosition(simGetObjectHandle("Left_leg_joint_6"),
math.pi/72+dist)
else
case=2
end
end

if(case==2) then
if(simGetJointTargetPosition(simGetObjectHandle("Right_leg_joint_4"))<0.6)
then
simSetJointTargetPosition(simGetObjectHandle("Right_leg_joint_4"),
math.pi/24+dist)
simSetJointTargetPosition(simGetObjectHandle("Right_leg_joint_3"), -
math.pi/12-dist)
else
case=3
end
end


if(case==3) then
if(simGetJointTargetPosition(simGetObjectHandle("Right_leg_joint_5"))<0.09)
then
simSetJointTargetPosition(simGetObjectHandle("Right_leg_joint_5"), -
(math.pi/6-dist))
simSetJointTargetPosition(simGetObjectHandle("Right_leg_joint_6"), 0)
end
end


end
```

Appendix G: VREPROBOT function in Matlab for 2 DOF.

```
function [out]=VREPROBOT(in)
qdr1=in(1);
qdr2=in(2);

vrep=remApi('remoteApi'); %with the prototype file (remoteApiProto.m)
vrep.simxFinish(-1);%close all connections before
clientID=vrep.simxStart('127.0.0.1',19999,true,true,5000,5);

%%Handle joints:
[r1,
j1]=(vrep.simxGetObjectHandle(clientID,'Joint_1#',vrep.simx_opmode_blocking))
;%%r1=a remote API function return code////j1=position
[r2,
j2]=(vrep.simxGetObjectHandle(clientID,'Joint_2#',vrep.simx_opmode_blocking))
;

%%Move joints to the reference velocity:
vrep.simxSetJointTargetVelocity(clientID, j1, qdr1,
vrep.simx_opmode_oneshot_wait);
vrep.simxSetJointTargetVelocity(clientID, j2, qdr2,
vrep.simx_opmode_oneshot_wait);

%%Get the real position:
[s1, q1]=vrep.simxGetJointPosition(clientID, j1,
vrep.simx_opmode_oneshot_wait);
[s2, q2]=vrep.simxGetJointPosition(clientID, j2,
vrep.simx_opmode_oneshot_wait);

%%CHANGE THE FORMAT:
q1d=double(q1);
q2d=double(q2);

out=[q1d, q2d];
end
```

Appendix H: Discrete PID controller function in Matlab for 2 inputs.

```
function [out] = PID_pos(in)
e1k=in(1);
e2k=in(2);
t=in(3);

persistent int_e1k_1 int_e2k_1


%Variables inialitation:
if(t<1e-8)
    int_e1k_1 =0; int_e2k_1 =0;
end

T=0.01;

%PID parameters:
Ti=[1e20; 1e20];
Td=[0; 0];
Kp=[0.2; 0.2];

e1vk=e1k/T;
e2vk=e2k/T;

%Sum of errors
int_e1k= int_e1k_1 + e1k*T;
int_e2k= int_e2k_1 + e2k*T;

%Output motor current:
qd1r=Kp(1)*(e1k+Td(1)*e1vk+1/Ti(1)*int_e1k);
qd2r=Kp(2)*(e2k+Td(2)*e2vk+1/Ti(2)*int_e2k);

%Variables update:
int_e1k_1= int_e1k;
int_e2k_1= int_e2k;

qd1rd=double(qd1r);
qd2rd=double(qd2r);

[out]=[qd1rd, qd2rd];
end
```

Appendix I: Joint position reference function in Matlab for dynamic control.

```
function [out] = qref(in)
t=in(1);

persistent q1_1 q2_1;

if(t<1e-8)
    q1_1=0; q2_1=0;
end

if(t>1e-8 && q2_1<pi/4)
        q1r=q1_1+0.1;
        q2r=q2_1+0.1;

else
       q1r=q1_1;
       q2r=q2_1;
end

%%Variables update
        q1_1=q1r;
        q2_1=q2r;

out=[q1r, q2r];

end
```

Appendix J: Pseudoinverse function in Matlab for 2 DOF robot.

```
function [out]=PSEUDO_2GOF (in)
q1=in(1);
q2=in(2);
xd=in(3);
yd=in(4);
zd=in(5);


L1=5e-1;
L2=5e-1;

v=[xd; yd; zd];

J=[- L2*sin(q1 + q2) - L1*sin(q1), -L2*sin(q1 + q2);
   L2*cos(q1 + q2) + L1*cos(q1),  L2*cos(q1 + q2);
                              0,                 0];
pseudo=pinv(J);

qd=pseudo*v;

out=qd;
end
```

Appendix K: Forward Kinematics function in Matlab for 2 DOF robot.

```
function [out] = FK_2GOF(in)

q1=in(1);
q2=in(2);

L1=5e-1;
L2=5e-1;


x = L1*cos(q1)+L2*cos(q1+q2);
y = L1*sin(q1)+L2*sin(q1+q2);
z=0;

out=[x, y, z];
end
```

Appendix L: XYZ Circle path in Matlab.

```
function [out] = circle_path(in)
t=in(1);

persistent angx_1 angy_1;
angx=0; angy=0;
if(t<1e-8)
    angx_1=0; angy_1=0;
    angx=0; angy=0;
end

if(t>1e-8)
        angx=angx_1+0.01;
        angy=angy_1+0.01;
        xr=cos(angx);
        yr=sin(angy);
        zr=5e-2;

else
        xr=cos(angx_1);
        yr=sin(angy_1);
        zr=5e-2;
end

%%Variables update
       angx_1=angx;
       angy_1=angy;

out=[xr, yr, zr];

end
```

Appendix M: DLS function in Matlab for 4 DOF robot.

```
function [out]=DLS_4GOF (in)
q1=in(1);
q2=in(2);
q3=in(3);
q4=in(4);
xd=in(5);
yd=in(6);
zd=in(7);
k=in(8);
E=in(9);

L=5e-1;
v=[xd; yd; zd];

J =[ -L*(sin(q1 + q2 + q3) + sin(q1 + q2 + q3 + q4) + sin(q1 + q2) +
sin(q1)), -L*(sin(q1 + q2 + q3) + sin(q1 + q2 + q3 + q4) + sin(q1 + q2)), -
L*(sin(q1 + q2 + q3) + sin(q1 + q2 + q3 + q4)), -L*sin(q1 + q2 + q3 + q4);
  L*(cos(q1 + q2 + q3) + cos(q1 + q2 + q3 + q4) + cos(q1 + q2) + cos(q1)),
L*(cos(q1 + q2 + q3) + cos(q1 + q2 + q3 + q4) + cos(q1 + q2)),  L*(cos(q1 +
q2 + q3) + cos(q1 + q2 + q3 + q4)),  L*cos(q1 + q2 + q3 + q4);
                                                                        0,
0,                                               0,
0];

k=k0*exp(-det(J*J')/2*E^2);
pseudo=J'*inv(J*J'+k^2*eye(3));
qd=pseudo*v;

out=qd;
end
```

Appendix N: Forward Kinematics function in Matlab for 4 DOF robot.

```
function [out] = FK_4GOF(in)

q1=in(1);
q2=in(2);
q3=in(3);
q4=in(4);

L=5e-1;


x = L*(cos(q1 + q2 + q3) + cos(q1 + q2 + q3 + q4) + cos(q1 + q2) + cos(q1));
y = L*(sin(q1 + q2 + q3) + sin(q1 + q2 + q3 + q4) + sin(q1 + q2) + sin(q1));
z=0;

out=[x, y, z];
end
```

Appendix O: Peter Corke 1 DOF Robot

```
startup_rvc
%Parámetros D-H
R(1)=Link([0 0 0.5 0]);
R

%Parámetros dinámicos
R(1).m=4.335;
R(1).r=[-0.25 0 0];
R(1).I=4.335*[2.912e-01 0 0; 0 2.649  0; 0 0 2.781];
R(1).B=6.9589;%%Viscosidad
R(1).G=70;%%Relación de transmisión
R(1).Jm=0.0765;%%Inercia del motor


Robot_1DOF= SerialLink(R, 'name', '1DOF')

Robot_1DOF.gravity=[0 0 9.81]';

%%Representarlo:
Robot_1DOF.teach()
```

Appendix P: Peter Corke 2 DOF Robot

```
startup_rvc
%Parámetros D-H
R(1)=Link([0 0 0.5 0]);
R(2)=Link([0 0 0.5 0]);
R

%Parámetros dinámicos
R(1).m=4.335;
R(1).r=[-0.25 0 0];
R(1).I=4.335*[2.912e-01 0 0; 0 2.649 0; 0 0 2.781];
R(1).B=6.9589;%%Viscosidad
R(1).G=70;%%Relación de transmisión
R(1).Jm=0.0765;%%Inercia del motor


R(2).m=1.951;
R(2).r=[-0.25 0 0];
R(2).I=1.951*[1.242e-01 0 0; 0 2.145 0; 0 0 2.145];
R(2).B=6.9589;%%Viscosidad
R(2).G=70;%%Relación de transmisión
R(2).Jm=0.0765;%%Inercia del motor

Robot_2DOF= SerialLink(R, 'name', '2DOF')

Robot_2DOF.gravity=[0 0 9.81]';

%%Representarlo:
Robot_2DOF.teach()
```

Appendix Q: Biolid Kinect Control

```
-- Following function writes data to the socket (the data might be sent in
several packets)
writeSocketData=function(client,data)
    -- Check how many packets we need to send:
    local packetCount=0
    local s=#data
    while (s~=0) do
        packetCount=packetCount+1
        if (s>256-6) then -- this is the max packet size minus header size
            s=s-256+6
        else
            s=0
        end
    end
    -- Now send the data:
    s=#data
    local pointer=0
    while (s~=0) do
        packetCount=packetCount-1
        local sizeToSend=s
        if (s>256-6) then
            sizeToSend=256-6
        end
        s=s-sizeToSend
        local
header=string.char(59,57,math.mod(sizeToSend,256),math.floor(sizeToSend/256),
math.mod(packetCount,256),math.floor(packetCount/256))
        -- Packet header is: headerID (59,57), dataSize (WORD), packetsLeft
(BYTE)
        client:send(header..data:sub(pointer+1,pointer+sizeToSend))
        pointer=pointer+sizeToSend
    end
end

-- Following function reads data from the socket (that might be arriving in
several packets)
readSocketData=function(client)
    local returnData=''
    while (true) do
        -- Packet header is: headerID (59,57), dataSize (WORD), packetsLeft
(WORD)
        local header=client:receive(6)
        if (header==nil) then
            return(nil) -- error
        end
        if (header:byte(1)==59)and(header:byte(2)==57) then
            local l=header:byte(3)+header:byte(4)*256
            returnData=returnData..client:receive(l)
            if (header:byte(5)==0)and(header:byte(6)==0) then
                break -- That was the last packet
            end
        else
            return(nil) -- error
        end
    end
    return(returnData)
end
```

```
linkPoints=function(returnData,index1,index2,minConfidence)
    if
(returnData[4*index1+4]>minConfidence)and(returnData[4*index2+4]>minConfidenc
e) then
        local
data={returnData[4*index1+1],returnData[4*index1+2],returnData[4*index1+3],re
turnData[4*index2+1],returnData[4*index2+2],returnData[4*index2+3]}
        data[1]=data[1]-1
        data[4]=data[4]-1
        simAddDrawingObjectItem(lineContainer,data)
    end
end

threadFunction=function()
    while (simGetSimulationState()~=sim_simulation_advancing_abouttostop) do

        -- Send a request to the server (just anything):
        writeSocketData(client,' ')
        -- Read the reply from the server:
        local returnData=readSocketData(client)
        if (returnData==nil) then
            break -- Read error
        else
            returnData=simUnpackFloatTable(returnData)
            simAddDrawingObjectItem(lineContainer,nil)
            simAddDrawingObjectItem(sphereContainer,nil)
            torsoTransf=simGetObjectMatrix(objectHandle,-1)
            if (returnData[60]>0.5) then

torsoPos={returnData[57]*scalingFact/1000,returnData[58]*scalingFact/1000,ret
urnData[59]*scalingFact/1000}
                torsoPos=simMultiplyVector(m,torsoPos)
            end
            for i=0,15,1 do
                if (i<6)or(i>13) then
                if (returnData[4*i+4]>0.5) then

pointPos={returnData[4*i+1]*scalingFact/1000,returnData[4*i+2]*scalingFact/10
00,returnData[4*i+3]*scalingFact/1000}
                    pointPos=simMultiplyVector(m,pointPos)
                    pointPos[1]=pointPos[1]-torsoPos[1]
                    pointPos[2]=pointPos[2]-torsoPos[2]
                    pointPos[3]=pointPos[3]-torsoPos[3]
                    pointPos=simMultiplyVector(torsoTransf,pointPos)
                    returnData[4*i+1]=pointPos[1]
                    returnData[4*i+2]=pointPos[2]
                    returnData[4*i+3]=pointPos[3]
                    pointPos[1]=pointPos[1]-1
                    simAddDrawingObjectItem(sphereContainer,pointPos)

print(returnData[4*i+1],returnData[4*i+2],returnData[4*i+3])
                end
                end
            end
            linkPoints(returnData,0,2,0.5)
            linkPoints(returnData,1,3,0.5)
            linkPoints(returnData,5,14,0.5)
            linkPoints(returnData,4,14,0.5)
            linkPoints(returnData,2,4,0.5)
            linkPoints(returnData,3,5,0.5)
            linkPoints(returnData,4,5,0.5)
--             linkPoints(returnData,14,6,0.5)
--            linkPoints(returnData,14,7,0.5)
--            linkPoints(returnData,6,8,0.5)
--            linkPoints(returnData,7,9,0.5)
--            linkPoints(returnData,8,10,0.5)
--            linkPoints(returnData,10,12,0.5)
--            linkPoints(returnData,9,11,0.5)
--            linkPoints(returnData,13,11,0.5)
            linkPoints(returnData,4,15,0.5)
            linkPoints(returnData,5,15,0.5)

            if
(returnData[4*0+4]>0.5)and(returnData[4*1+4]>0.5)and(returnData[4*2+4]>0.5)an
d(returnData[4*3+4]>0.5)and(returnData[4*4+4]>0.5)and(returnData[4*5+4]>0.5)a
nd(returnData[4*14+4]>0.5) then
                pt1={returnData[4*4+1],returnData[4*4+2],returnData[4*4+3]}
                pt2={returnData[4*5+1],returnData[4*5+2],returnData[4*5+3]}

pt3={returnData[4*14+1],returnData[4*14+2],returnData[4*14+3]}
                v1={pt1[1]-pt2[1],pt1[2]-pt2[2],pt1[3]-pt2[3]}
                v2={pt3[1]-pt2[1],pt3[2]-pt2[2],pt3[3]-pt2[3]}
                n={v1[2]*v2[3]-v1[3]*v2[2],v1[3]*v2[1]-
v1[1]*v2[3],v1[1]*v2[2]-v1[2]*v2[1]}
                l=math.sqrt(n[1]*n[1]+n[2]*n[2]+n[3]*n[3])
                n[1]=n[1]/l
                n[2]=n[2]/l
                n[3]=n[3]/l
        dd={0,0,1.5,n[1],n[2],1.5+n[3]}


                correctionAngle=math.asin(n[3])
--        simAddDrawingObjectItem(lineContainer,dd)

                z={v1[2]*n[3]-v1[3]*n[2],v1[3]*n[1]-v1[1]*n[3],v1[1]*n[2]-
v1[2]*n[1]}
                l=math.sqrt(z[1]*z[1]+z[2]*z[2]+z[3]*z[3])
                z[1]=z[1]/l
                z[2]=z[2]/l
                z[3]=z[3]/l
        dd={0,0,1.5,z[1],z[2],1.5+z[3]}
--        simAddDrawingObjectItem(lineContainer,dd)

--                n[1]=-n[1]
--                n[2]=-n[2]
--                n[3]=-n[3]

                x={n[2]*z[3]-n[3]*z[2],n[3]*z[1]-n[1]*z[3],n[1]*z[2]-
n[2]*z[1]}

        dd={0,0,1.5,x[1],x[2],1.5+x[3]}
--        simAddDrawingObjectItem(lineContainer,dd)

                lsp=simGetObjectPosition(leftShoulder,-1)
                rsp=simGetObjectPosition(rightShoulder,-1)

                mm={0,0,0,0,0,0,0,0,0,0,0,0}
                mm[1]=x[1]
                mm[2]=n[1]
                mm[3]=z[1]
                mm[4]=returnData[17]
                mm[5]=x[2]
                mm[6]=n[2]
                mm[7]=z[2]
                mm[8]=returnData[18]
                mm[9]=x[3]
                mm[10]=n[3]
                mm[11]=z[3]
                mm[12]=returnData[19]

                mml=simGetInvertedMatrix(mm)
                mm[4]=returnData[21]
                mm[8]=returnData[22]
                mm[12]=returnData[23]
                mmr=simGetInvertedMatrix(mm)


leftHandP=simMultiplyVector(mml,{returnData[1],returnData[2],returnData[3]})
--        leftHandP={0,0,0}
                leftHandP[1]=leftHandP[1]+lsp[1]
                leftHandP[2]=leftHandP[2]+lsp[2]
                leftHandP[3]=leftHandP[3]+lsp[3]


rightHandP=simMultiplyVector(mmr,{returnData[5],returnData[6],returnData[7]})
--        rightHandP={0,0,0}
                rightHandP[1]=rightHandP[1]+rsp[1]
                rightHandP[2]=rightHandP[2]+rsp[2]
                rightHandP[3]=rightHandP[3]+rsp[3]


leftElbowP=simMultiplyVector(mml,{returnData[9],returnData[10],returnData[11]
})
--        leftElbowP={0,0,0}
                leftElbowP[1]=leftElbowP[1]+lsp[1]
                leftElbowP[2]=leftElbowP[2]+lsp[2]
                leftElbowP[3]=leftElbowP[3]+lsp[3]


rightElbowP=simMultiplyVector(mmr,{returnData[13],returnData[14],returnData[1
5]})
--        rightElbowP={0,0,0}
                rightElbowP[1]=rightElbowP[1]+rsp[1]
                rightElbowP[2]=rightElbowP[2]+rsp[2]
                rightElbowP[3]=rightElbowP[3]+rsp[3]

            simSetObjectPosition(leftHand,-1,leftHandP)
            simSetObjectPosition(rightHand,-1,rightHandP)

            simSetObjectPosition(leftElbow,-1,leftElbowP)
            simSetObjectPosition(rightElbow,-1,rightElbowP)

            end


        end
        simSwitchThread()
    end
end

simSetThreadSwitchTiming(200) -- We wanna manually switch for synchronization
purpose (and also not to waste processing time!)

-- We start the server on a port that is probably not used (try to always use
a similar code):
simSetThreadAutomaticSwitch(false)
```

```
local portNb=simGetIntegerSignal('freeLocalServerPort',true)
local portStart=simGetInt32Parameter(sim_intparam_server_port_start)
local portRange=simGetInt32Parameter(sim_intparam_server_port_range)
if (not portNb) then
    portNb=portStart
end
local newPortNb=portNb+1
if (newPortNb>=portStart+portRange) then
    newPortNb=portStart
end
simSetIntegerSignal('freeLocalServerPort',newPortNb,true)
simSetThreadAutomaticSwitch(true)
simLaunchExecutable('kinectServer.exe',portNb,1)

-- Build a socket and connect to the server:
 socket=require("socket")
 client=socket.tcp()
simSetThreadIsFree(true) -- To avoid a bief moment where the simulator
appears as locked
local result=client:connect('127.0.0.1',portNb)
simSetThreadIsFree(false)

-- Prepare the drawing containers for lines and spheres (to display the
skeleton):
lineContainer=simAddDrawingObject(sim_drawing_lines,4,0,-1,100,{1,0,0})
sphereContainer=simAddDrawingObject(sim_drawing_spherepoints,0.05,0,-
1,100,{0,0,1})
objectHandle=simGetObjectHandle('Biolid')
leftHand=simGetObjectHandle('leftHandSphere')
rightHand=simGetObjectHandle('rightHandSphere')
leftElbow=simGetObjectHandle('leftElbowSphere')
rightElbow=simGetObjectHandle('rightElbowSphere')
leftShoulder=simGetObjectHandle('leftShoulderSphere')
rightShoulder=simGetObjectHandle('rightShoulderSphere')
torsoPos={0,0,0}
scalingFact=0.7
w=0.7
correctionAngle=0
m=simBuildMatrix({0,0,0},{math.pi/2,0,0})
if (result==1) then
    -- Here we execute the regular thread code:
    res,err=xpcall(threadFunction,function(err) return debug.traceback(err)
end)
    if not res then
        simAddStatusbarMessage('Lua runtime error: '..err)
    end
end
client:close()
```