University of Liège - Faculty of engineering

Master thesis

Simulation of complex actuators

Author : Hubert Woszczyk
Promotor : Pr. Bernard Boigelot

Graduation Studies conducted for obtaining the
Master's degree in Electrical Engineering
by Hubert Woszczyk

Academic year 2015-2016

Simulation of complex actuators

Hubert Woszczyk

Abstract

Lorem ipsum dolor...

Acknowledgements

I would like to express my sincere thanks to all those who provided me the possibility to complete this thesis.

First of all, I thank the professor B. Boigelot who made this thesis possible and helped me on various occasions.

I also wish to thank Grégory Di Carlo and Guillaume Lempereur, my fellow students who also worked on the robot.

Contents

List of Figures

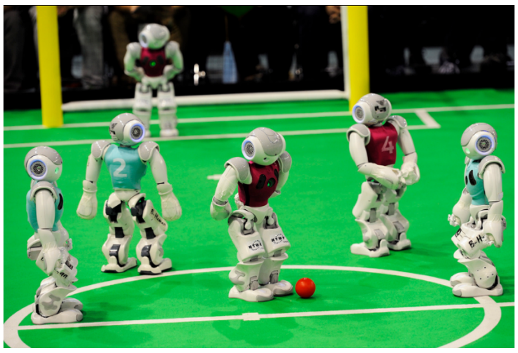
List of Tables

Chapter 1

Introduction

1.1 Context

For the last ten years, students from the Montefiore institute have been participating in a robotic contest named *Eurobot*, a competition in which wheeled robots battle each other for points in various play environments. After some success and following a thirst for new challenges, it was decided to move on to another contest, *RoboCup*.


(a) Two teams of Nao robots playing against each other in the 2014 edition of RoboCup Soccer standard platform league. *[Photo courtesy of RoboCup]*

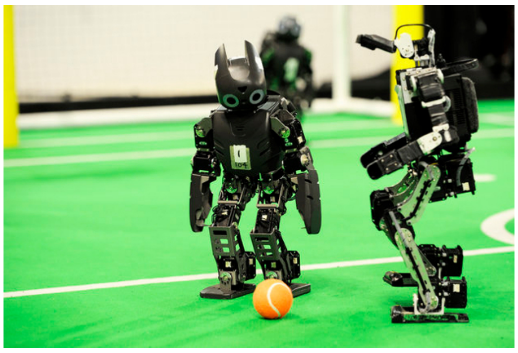

(b) Two robots of opposing teams looking at the ball, in the 2013 edition of RoboCup Soccer kidsize league. *[Photo courtesy of RoboCup]*

Figure 1.1: Robocup standard and kidsize leagues

This contest is quite vast and, as of 2016, is divided into several categories (called domains in RoboCup jargon):

- RoboCup Rescue : as the name suggests, a domain where robots must perform various rescue operations in diverse scenarios.
- RoboCup Industrial : a category with industrially oriented competitions,
- RoboCup@Home : centred around domestic robots, such as robotics helpers for the elderly or robotic butlers.
- RoboCupJunior : more of an initiative that aims to foster robotics interest in children rather than a contest, it helps organize various robotics events for younger minds.

– RoboCup Soccer : historically the first category, centred about humanoid robots playing football. The objective of this category is to have a team of robots beat the world champions by 2050. This is the category we will compete in.

RoboCup Soccer is further subdivided into 4 sub-categories called leagues :

– Standard platform, where the teams all use the same robot, *Nao*, as illustrated in Figure 1.1a.

– Simulation, a league that does not feature physical robots but focuses on team strategies and artificial intelligence. The matches take place in 2D or 3D simulators.

– Adultsize, for the taller robots.

– Teensize, for middle sized robots.

– Kidsize, for the smaller robots. Figure 1.1b shows robots from that league.

This year's team is preparing to participate to the Kidsize league and this master thesis, along with two others, is the by-product of that team's activity. Since this is our first time participating we have no experience regarding humanoids robots. To avoid spending countless hours building and testing different designs we need a tool able of simulating the physics of a robot model.

1.2 Goals of the project

The goal of this thesis is to provide the team with a physics simulating tool with the following features :

– realistic simulation of the physics of rigid bodies. This means that the tool should handle inertia, collisions, friction and constraints between objects. Simulation of springs and dampers is an interesting bonus.

– receive and process orders incoming at a relatively high frequency. The processing need not be in real-time.

– the model of our robot should receive the same orders as the real robot would. That is, the simulator should provide the same interface to the control code as the real robot would.

– 3D visualization of the simulation.

That simulator will be used to :

1. Test different robot designs and choose the best one, in a more efficient way than it could be achieved by physically building the designs.

2. Speed up development and testing of the control code because multiple teams will be able to work in parallel.

1.3 Structure of the report

This report begins with an overview of the basic concepts behind physics simulation on computers in chapter 2. We then move on to the chapter that motivates the choice of V-Rep as the main simulation tool for this project.

Chapter 4 will be about the modelling of the build blocks of our humanoid robot. The verification and the tuning of the results of some basic simulations will be made in chapter 5.

Chapter 6 goes into the core of the subject with some simulations that influenced the design of the robot before being used to explore control strategies. The last chapter will conclude the work by summing up and laying out future prospects.

Chapter 2

Principles of interactive rigid body simulation

This chapter briefly introduces the basic concepts of physics engines. We restrict ourselves to the simulation of rigid bodies, which is a significant simplification of the problem since rigid bodies are idealized solid objects which never change shape.

2.1 Problem statement

Physics engine trouble themselves with the simulation of classical mechanics in a computer. They model how objects accelerate, move and react to collisions with other objects. They also model how objects can be constrained to each other, for example with a hinge and how that influences them.

2.1.1 Notations and definitions

– a denotes a scalar.

– $\mathbf{b}$ denotes a vector.

– $\cdot$ denotes a product.

– $\times$ denotes a cross product.

– A **mesh** is a 3D object made of vertices, edges and faces.

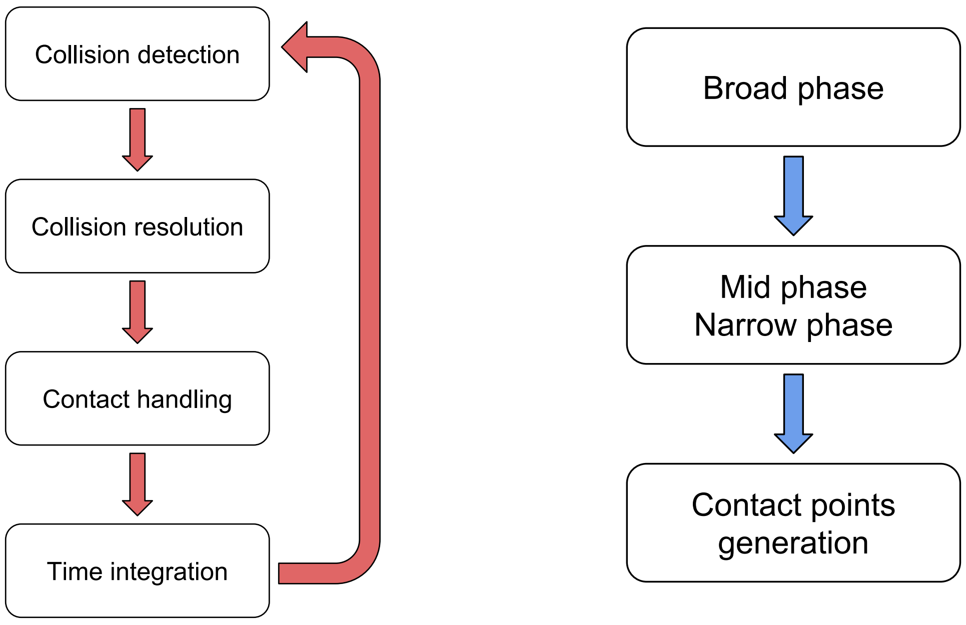
– A **convex mesh** is a mesh whose internal angles are all less or equal to 180°.

2.2 Principles of rigid body dynamics simulation

The section is heavily inspired by Bender's [**?**] state of the art paper on rigid body simulation. The simulation of rigid body dynamics is usually built around the loop presented in fig. 2.1a. The simulator begins by finding the collision points between objects (Collision detection). These points are used to derive motion laws which are solved to determine the forces that act on the objects and prevent them from inter-penetrating (Contact handling). Newly found contact points imply collisions, which generate infinite impulse forces, which are handled by collision resolution. When all the contact forces have been computed, the position and velocities of the bodies are integrated forward in time before a new iteration starts.

Rigid body simulation is achieved through the expression of the Newton-Euler laws as differential equations which are then augmented with equations that express 3 conditions : nonpenetration of bodies, the friction model, and certain disjunctive relationships between variables (a contact force must become zero if two bodies separate, the friction forces acts in the direction that will most quickly stop the sliding).

This yields a differential nonlinear complementarity problem (dNCP) that cannot be solved in closed form. It is discretized in time producing a series of NCPs whose solutions are an approximation of the state of the system. This discrete solution is usually found by linearizing the NCP into a LCP to take advantage of the rich background for that type of problems.



(a) Modular description of the simulation loop of a physics engine

(b) Modular description of the collision detection in a physics engine.

Figure 2.1: Modular phase description of the sub tasks of a rigid body simulator. The mid phase and narrow phase are grouped together because they are often combined for performance reasons

2.3 Collision detection

Collision detect is broken into three phases called the broad phase, the mid phase and the narrow phase.

During the broad phase, objects are approximated by simple geometric primitives.

Distances between such geometric shapes are easy to compute. Spheres are usually used. If such spheres do not overlap, then neither do the actual objects.

When an object has a complex shape, an additional phase called the mid phase separates the object into several simpler shapes to detect collisions. Finally the narrow phase uses the exact geometries of the object to find the contact points. These are then returned to the simulation model.

2.4 Formulating the nonlinear complementarity problem (physics model)

2.4.1 Kinematics

The position of an object in a 3D space is given by a vector $p \in R^3$ from the origin of an inertial frame to the body fixed frame.

The orientation of an object in a 3D space can be represented in different ways. Usually it represented by either Euler angles or unit quaternions (Q_s, Q_x, Q_y, Q_z)

The translational velocity is usually noted $\mathbf{v} \in \mathcal{R}^3$. The rotational velocity $\mathbf{w} \in \mathcal{R}^3$ describes the rate at which the body rotates.

If $\mathbf{q} = (p, Q)^T$ the differential equation of motion can be written as

$$\dot{\mathbf{q}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -Q_x & -Q_y & -Q_z \\ 0 & 0 & 0 & Q_s & Q_z & -Q_y \\ 0 & 0 & 0 & -Q_z & Q_s & Q_x \\ 0 & 0 & 0 & Q_y & -Q_x & Q_s \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

2.4.2 Newton-Euler

The Newton-Euler equations describe how forces and moments modify the velocities of an object.

$$m\dot{\mathbf{v}} = \mathbf{f} \tag{2.1}$$

$$\mathbf{I}\dot{\omega} + \omega \times \mathbf{I}\omega = \tau \tag{2.2}$$

Chapter 3

Choosing the tools

In this chapter we present the problem we want to solve with the simulator and present some of the most popular simulation tools we surveyed. We finally choose one that fits our needs.

3.1 Problem statement

Our robot will consist a number of servos connected together through frames in addition to a central plate that will contain the electronics. We also plan to use spring dampers in the legs to mimic human movement more.

If we want to simulate it we thus need a tool that handles:

- inertia, to have physically accurate dynamics.
- joints to have constraints between objects.
- collision and friction, to accurately model the interaction of the feet with the ground.
- spring dampers.

The robot will also have several captors such as accelerometers, rotary position encoders in the servos and we also want to model them in the simulator as they are used by the control algorithms.

We will also have cameras on the robot, therefore if we want to take a holistic approach we should try to model them as well but it is not a priority.

The final requirement is that we want to be able to control the robot from outside the simulator.

3.2 Barebone physics engines

The physics engine is the cornerstone of a physics simulation tool. There exist a quantity of them, we present here the most popular ones that have the the following features' set : multi rigid body dynamics, collision detection, joints (hinge, spring-dampers, generic 6DOF[1]).

1. **Bullet[2]:** As of now, the most popular open source physics engine. Although primarily used in video games it is also used in applications such as Blender, V-Rep or NASA's tensegrity robotics toolkit.

2. **Newton[3]:** Another open source engine, not quite as popular as Bullet and ODE but nevertheless used for commercial games and simulation.

3. **ODE[4]:** Open source, a little older that Bullet. As it was already mature when simulators started to be developed, it is present in a lot of robotics simulation tools (V-Rep, Webots, Gazebo...). It was also designed with games in mind but was influenced by its success in more serious applications.

4. **PhysX & Havok:** Both are proprietary engines used primarily for games. They won't be further discussed because their focus is on speed rather than accurateness and as such they cannot be used in a simulation that aims to be realistic, as stated by Erez *et al* in [**?**].

3.3 Simulators

In this section we will discuss software that provides a higher level interface to the physics engines we presented earlier. That interface usually adds a visualization and some other useful features.

1. **Blender[5]** is 3D modelling software suite and as such has integrated the Bullet engine to help make more realistic animations. It features the ability to make Python scripts that use that engine to make games or physics simulations.

   It is open source and cross platform.

2. **Gazebo[6]** was the official simulator of DARPA's Robotics challenge. It features multiple physics engines (Bullet ,Simbody, Dart and ODE), allows custom plugins and uses the SDF format for its models.

   It is open source and runs natively on Linux systems but needs to be compiled in order to run on Windows or OSX.

3. **V-Rep[7]** is another simulator that lets you choose the physics engine(Bullet, ODE, Newton, Vortex) and it also allows custom plugins in the form of LUA scripts. It uses its own format for storing models but can import standard formats(COLLADA, 3ds, etc...).

   It is cross-platform and free to use for educational purposes.

4. **Webots[8]** has virtually the same features as V-Rep.

[1]DOF : degree of freedom
[2]http://bulletphysics.org/wordpress/
[3]http://newtondynamics.com/forum/newton.php
[4]https://bitbucket.org/odedevs/ode/
[5]https://www.blender.org/ [Accessed 21/05/2016]
[6]http://gazebosim.org/ [Accessed 21/05/2016]
[7]http://www.coppeliarobotics.com/ [Accessed 21/05/2016]
[8]https://www.cyberbotics.com/ [Accessed 21/05/2016]

It is cross platform but not free.

5. **Matlab** is not a dedicated robotics simulator *per se* but can be used to model the robot analytically and to write simulation code for it.

A summary of the features of each simulator is present on table 3.1.

Simulator	**License**	**Physics engine(s)**	**Integrated editor**	**Modelling**
Blender	Free	Bullet	Fully fledged	Internal
V-REP	Free	Bullet, ODE, Newton, Vortex(10s limit)	Limited	Can import .COLLADA
Gazebo	Free	Bullet, ODE, Simbody, DART	Limited	SDF format
Webots	Proprietary	ODE	None	SDF format
Matlab	Proprietary	None	None	Mathematical

Table 3.1: Comparison of simulators

3.4 Tested software

In this section we try some of the proposals presented previously and give our thoughts on them. We will mainly look at the modelling facilities and the access to the underlying simulation options as all the proposed tools possess the required simulation capabilities and are able to deliver similar looking results.

3.4.1 Blender

Blender is convenient to use because the robot's model can be easily changed inside it and the fact that the scripting language is Python make for faster development. Support for a socket allows an external program to control the simulation and the robot inside it. The internals of the physics are obscured and some interesting object properties, such as inertias, are hard to reach. It is also hard to change the simulation parameters making it difficult to obtain stable results when using a higher number of objects and constraints. Furthermore, support for the game engine, the basis of a simulation project, is uncertain, as stated in this development [?].

Pros :

– Easy modelling of the elements.

– The Python API is well documented.

Cons :

– Bullet's simulation parameters are hidden behind an incomplete interface, making it impossible to modify the timestep or the number of iterations for constraint solving.

– Friction parameter not a physical value but a value between 0 and 1.

– Inertias are approximated by the principal values Ixx, Iyy and Izz.

3.4.2 Gazebo

Gazebo is attractive because it has the support of DARPA and handles multiple physics engines. The main drawback lies in the modelling of the robot to be simulated. It does feature an internal modelling tool but it is too limited to be usable. That would not be a problem if it could import models easily but that is not the case, it uses an xml file to store the parameters of the robot and the only tool that can export models to that format is 3ds max, a commercial product. The team behind Gazebo seems to be well-aware that this is an issue since as of May 2015 it is focusing on developing an internal model editor.

Pros :

– Gives the choice of the physics engine.

– Inertias definable as a matrix.

– Friction represented in physical values.

Cons :

– Internal model editor prohibitively limited, cannot be used to create a complicated model. It cannot import popular CAD formats.

– Uses the SDF[9] format to store models, making it hard to iterate over robot designs.

3.4.3 V-Rep

V-Rep also has multiple physics engines available and has a user-friendly interface. It also has an internal modelling tool but there is not much use for it since it allows the import of models in the COLLADA format. It also supports socket communication and even provides code for a client thread in the custom application. The options of the physics engines are also pretty accessible and lots of sensor types are natively supported by the simulator.

Pros :

– Gives the choice of the physics engine.

– Inertias correctly definable.

– Friction represented in physical values.

– Already used at the university.

Cons :

– Limited internal editor, can hardly be used for modelling.

[9]A XML file

3.5 Choice

3.5.1 Simulator

The first choice to be made is whether we go for a barebone physics engines or a simulator: the former has the advantage of being a highly customizable solution but a simulator provides features a physics engines does not :

– 3D visualization

– code handling the import of models

– a remote API

All these functionalities would eventually be necessary so a simulator is preferred.

From the available simulators(Blender, Gazebo and V-Rep) the best seems to be V-Rep. This choice is further confirmed by Ivaldi in [?] who shows that V-Rep is the highest noted tools amongst roboticists.

Inside V-Rep, we chose Newton Dynamics as the physics engine because simple tests showed it to be the most stable with a high number of joints[10]. That choice is further confirmed by Hummel *et al* in [?] where Newton Dynamics is stated to be the best engine when it comes to handling a high number of constraints.

3.5.2 Modeller

Although Blender was not chosen as the primary simulation tool for the project, it shall be used as a modelling tool for the robot, as a complement for V-Rep.

[10]with the exception of Vortex but it requires a license to run more than $10s$

Chapter 4

Modelling the basic elements of a robot

This chapter covers the modelling of the building blocks of the robot.

4.1 Problem statement

Our robot will be made from a number of elements that all need to be represented in the simulation :

- servos
- cameras
- hands, feet
- electronics

We will first model the inactive elements such as the feet or the electronics before modelling the active elements, servos and cameras.

4.1.1 Convex elements

A convex element is an element whose interior angles are all less or equal to 180°. Our humanoid robot will have some number of them since a lot of elements can be approximated as cubes, which are convex.

The modelling consists in creating a mesh with the right dimensions, setting the mass and setting the inertia (V-Rep does not feature matrix inertias, only principal axes inertias). Friction of the material can also be set and will influence how much an object will slide.

4.1.2 Concave elements

A shape is concave if it is not convex. It is not recommended to use concave shapes in a simulation as they make collision detection more expensive and the simulation is generally more unstable.

Therefore, it is best to approximate such a shape by a convex mesh whenever possible. If not, then we separate it into several convex shapes that we group together with constraints.

4.2 Modelling the inactive elements

Module	**Weight** $[g]$	**Density** $[kg/m^3]$	**Dimensions** $[mm \times mm \times mm]$
Odroid C-2	40	840	85.0 x 56.0 x 10.0
Li-Po battery	188	2304	103.0 x 33.0 x 24.0
Mx-28R	72	1150	35.6 x 50.6 x 35.5
Hand			70.0 x 31.0 x 31.0
Feet			

Table 4.1: Weights and dimensions of the pieces of the robot. The density is useful for the automatic computation of the weight and inertia of the pieces in V-REP.

4.2.1 Electronics

By electronics we mean the motherboard and the power electronics. The motherboard is an Odroid C-2, shown in fig. 4.1. It is intended to be the sole CPU onboard the robot. The role of the power electronics will be to convey the energy stored in the batteries to the servo motors.

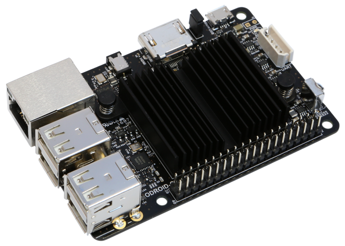


Figure 4.1: Odroid C-2 motherboard. Equipped with an ARM 2GHz quadcore CPU and 2GB of DDR3 RAM.

As of now, only the size of the motherboard is known to us, as the power electronics are not yet ready. Nevertheless, a rough guess can be made : we will assume that the electronics will take twice the volume of the Odroid C-2. That volume is represented as a cube, of dimensions and size compiled in table 4.1.

4.2.2 Hands

We call *hands* the elements that will mimic human hands. As the Robocup competition does not require to manipulate objects, the hands may well be simple tubes or cubes. However, in an effort to make the robot more versatile, grippers of some kind may be used instead.

We chose to model the hands as cylinders, of dimensions and size as presented in table 4.1.

4.2.3 Feet

At the moment of writing this report, the drawings of the feet are not completed. Nevertheless we can assume that approximating them as cube will not be too far from reality. The exact dimensions and weight are shown in table 4.1.

The choice of the material to use for the feet is rather important. The friction will have some influence on the behaviour of the robot. According to the material chosen, we can modify the friction parameter of the feet.

4.2.4 Central plate

The central plate will be the main body of the robot. It will support the electronics, the batteries and the arms and legs will be attached to it.

4.2.5 Frames

In an effort to make the simulation as fast and stable as possible, frames are not physically active in the simulation. They are present, but only for visualisation purpose and do not influence the outcome of the simulation.

4.2.6 Springs

It is planned to have springs on the legs of the robot, to try and make walking more energy efficient.

V-Rep provides an object that can represent a spring, the *prismatic joint*.

4.3 Modelling the active elements

4.3.1 Cameras

The robot will be equipped with two small cameras, of the type illustrated in fig. 4.2, used to locate the robot on the field[1].

We model the hull of the camera as cube. The dimensions and weight are in table 4.1. We model the active part of the camera as a *vision sensor*, a V-Rep object that simulates cameras. It can be customized to have the desired resolution, field-of-view, ...

That image sensor can be reached through the remote API of V-Rep in a number of ways :

[1]This is the subject of another master thesis this year

Figure 4.2: LI-USB30-M021C camera.

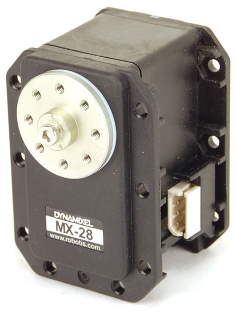
- Streaming : the data from the vision sensor can be sent continuously to the control code but this slows down the simulation significantly. V-Rep supports streaming, reducing the communication overhead.
- Oneshot : we can request one image from the simulator. This is less expensive and will probably be preferred.

All the characteristics of the model of a camera are compiled in table 4.2.

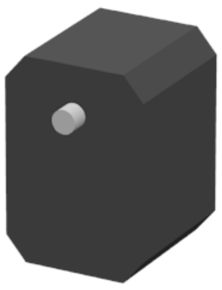
	Data	**Unit**
Weight	22	g
Dimensions	26.0 x 26.0 x 14.7	mm^3
Inertia		gmm^3
Resolution		px^2
Field of view		$^\circ$

Table 4.2: Characteristics of the model of the LI-USB30-M021C camera

4.3.2 Servos



(a) MX-28R servo.



(b) Model of the MX-28R.

Figure 4.3: Side by side of a MX-28R servo and its 3D model. The shape has been simplified but retains outer appearance of the servo. The axis is used as a position marker and will be removed once the joints are in place.

The robot will mainly be made from MX-28R servos (fig. 4.3a) manufactured by Dynamixel. Their size and power make them an good choice for a humanoid robot. The goal of this section is thus to reproduce as accurately as possible the behaviour of this servo in our simulation.

The MX-28R outer shape is convex so we create a convex mesh to model its appearance. As the manual [?] does not provide the actual continuous torque, we compute from the maximal torque of the DC motor and the reduction ratio of the gears.

$$\begin{aligned} Torque &= TorqueMotor \times ReductionRatio \\ &= 3.67e^{-3} \times 193 \\ &= 0.7083Nm \end{aligned}$$

The servo uses a PID controller to reach and hold a desired position. The model inside V-Rep also uses a PID and the code is in appendix C.

	Data	**Unit**
Weight	77	g
Dimension	$35.6 \times 50.6 \times 35.5$	mm^3
Inertia around main axes	$33,765 \quad 12,900 \quad 28,821$	$g \times mm^4$
Stall torque	2.5	Nm
Nominal torque	0.7	Nm

Table 4.3: Characteristics of a MX-28R type servo. Data taken from [?]

The original MX-28R servo does not have any accelerometry, but a master thesis in progress is going to change that. We can emulate it in the simulator by differentiating the position, obtained through *simxGetObjectPosition*.

Chapter 5

Physical validation

In this chapter experiments with real servos will be conducted in order to, firstly, tune the parameters of the simulation (servo's characteristics) and, secondly, verify that the simulation correctly predicts the behaviour of a real-life configuration.

5.1 Problem statement

Before using our simulator to test control algorithms it is useful to first verify that it gives physically accurate results. It would be a waste of time to conduct tests on a model that does not behave in the same way as the original does.

5.2 Experimental set-up

The set-up is explained on fig. 5.1. In later experiments a camera will be used to film the motion of the servos and compare it to the results of the simulation that is supposed to predict it.

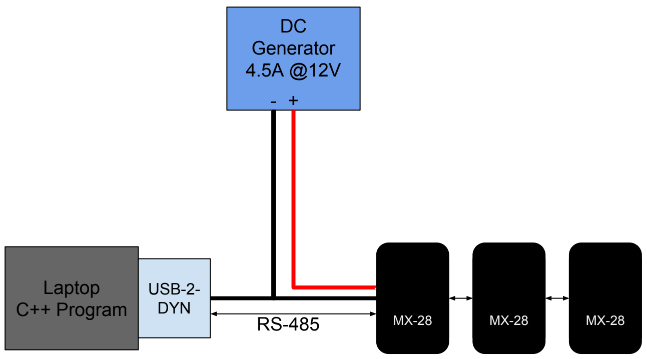


Figure 5.1: Experimental setup : The MX-28 servos are powered by a DC generator and controlled by a laptop equipped with a USB2DYNAMIXEL(USB-2-DYN) device. It converts an USB port into a serial port.

5.3 Experiment 1

The purpose of the first experiment is to test the torque : to that end, a frame is fixed onto a single servo and weighted. The setup is represented on fig. 5.2.

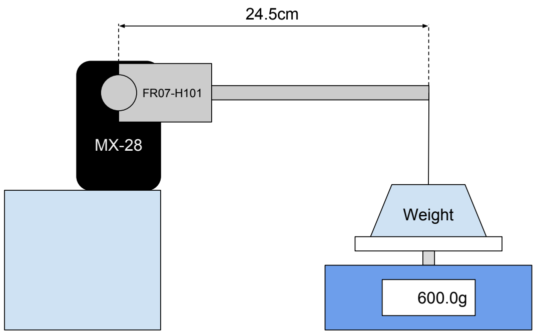


Figure 5.2: Experimental setup for torque testing. A weight w of is suspended at a distance d from the servo, resulting in a applied torque of $w \times g \times d$. The goal consists in finding the weight w for which the servo is unable to lift the arm.

In our case, d was equal to $22.5cm$ and we could reach a weight w of $740g$ at $14.8V$. This equals to a torque of $1.64Ncm$. The complete results are listed in table 5.1.

	Stall torque @11.1V $[N.m]$	**Stall torque @12V** $[N.m]$	**Stall torque @14.8V** $[N.m]$
Theoretical	2.1	2.5	3.1
Experimental			1.6

Table 5.1: Experimental stall torques at different tested voltages. Theoretical values taken from [?]

.

5.4 Experiment 2

In this experiment we will test some simple dynamics. The setup is on fig. 5.3.

5.5 Conclusion

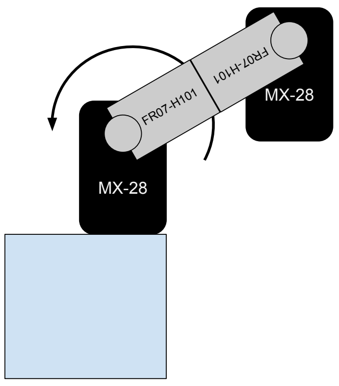


Figure 5.3: Experimental setup for dynamics testing. Two servos are connected together

Chapter 6

Applications

In this chapter we explain how to use the simulator and it influenced the design of our humanoid robot.

6.1 Overview of the simulation setup

V-Rep is used to simulate the physics of the robot but the control code runs alongside and not inside V-Rep. This is possible because V-Rep runs a server thread which can process specific instructions[1] sent by a client thread. This gives us great implementation flexibility and we can substitute the real robot by the simulation model easily. This is represented on fig. 6.1.

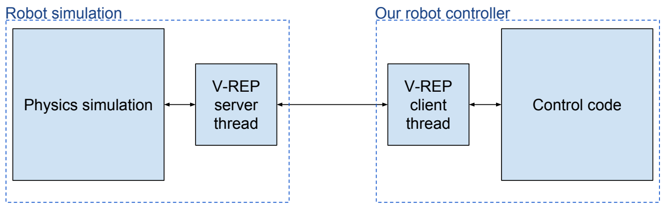


Figure 6.1: V-Rep simulates the robot while an external program sends order to the robot over TCP/IP thanks to the client/server thread provided by V-Rep.

Furthermore, the simulation will operate in the synchronous operation mode. That is, each simulation timestep must be triggered by the control code, allowing precise control of the robot. Figure 6.2 presents how V-Rep and the control code interact in synchronous mode.

The instructions available can execute a number of different actions. The following proved most useful for this project :

– simxGetObjectHandle : this function is used to retrieve a handle on an object. A handle is necessary if a user wants to perform operations on an object.

– simxSetJointTargetPosition : this function sets a target position for a joint.

[1]An alphabetical list of those instructions can be found on http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionListAlphabetical.htm

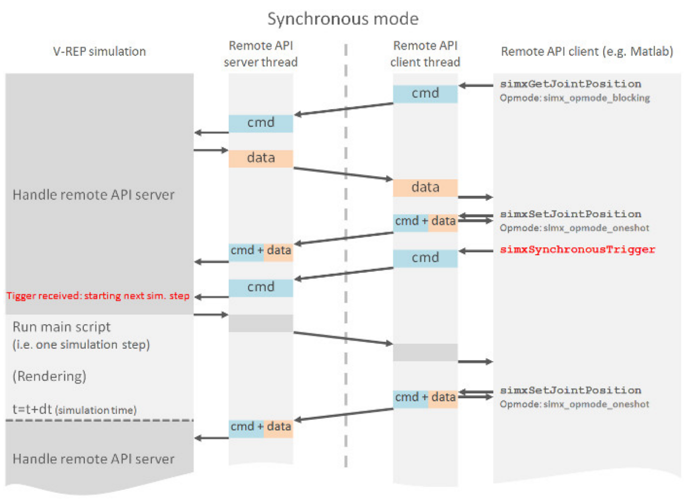


Figure 6.2: Typical interaction between the simulator and the control code. The simulation runs on two threads : the simulation and the server thread. The server threads can receive orders from a client thread which is controlled by a custom application of our own. The simulator waits for a trigger before simulating the next timestep.

- simxSetFloatSignal : this function gives the possibility of setting the value of a signal inside V-Rep. This is useful if we want to extend the interface that V-Rep provides.
- simxGetFloatSignal : this function retrieves the value of a float signal.

6.2 Applications

6.2.1 Static stability

The first application is simply to build a model of the robot and test if it is able to stand upright on its own, using the servos.

The torque of the servos is computed from the maximal torque of the DC motor and the reduction ratio of the gears.

$$\begin{aligned} Torque &= TorqueMotor \times ReductionRatio \\ &= 3.67e^{-3} \times 193 \\ &= 0.7083Nm \end{aligned}$$

In section 5.3 we determined that the maximum torque the servo was actually able to produce was $1.7Nm$ so to represent this we choose to set the maximum torque of the servos to $1Nm$.

In V-Rep the different elements of the robot are dynamically enabled and given mass, accordingly to the values listed in table 4.1. Then, joints (motor controlled with control loop activated) are added to simulate the behaviour of the servos. Their maximal torque is set to 1.6, the maximum torque developed my Mx-28 servos as shown by our earlier experiments (table 5.1).

```matlab
function simulation_client_vrep()

disp('Program started');
vrep = remApi('remoteApi'); % use the prototype file
vrep.simxFinish(-1); % close all opened connections
clientID = vrep.simxStart('127.0.0.1', 19997, true, true, 5000, 5);

if clientID < 0
    disp('Failed connecting to remote API server. Exiting.');
    vrep.delete();
    return;
end
disp('Connected to remote API server');

% Make sure we close the connexion whenever the script is interrupted.
cleanupObj = onCleanup(@() cleanup_vrep(vrep, clientID));

% Set the remote mode to 'synchronous'
vrep.simxSynchronous(clientID, true);

% retrieve handles to servos, joints
h = robot_init(vrep, clientID);

% start the simulation
vrep.simxStartSimulation(clientID, vrep.simx_opmode_oneshot_wait);

t = 0;
dt = 0.1; %timestep of the simulation
while true && t < 3
    instructions = standup_prone(h, t);
    send_instructions(vrep, clientID, instructions);
    t = t + dt;
end

% Before closing the connection to V-REP, make sure that the last command sent out had time to arrive.
vrep.simxGetPingTime(clientID);

% Now close the connection to V-REP:
vrep.simxStopSimulation(clientID, vrep.simx_opmode_oneshot_wait);
vrep.simxFinish(clientID);
vrep.delete(); % call the destructor!
disp('Program ended');
end
```

Listing 6.1: Minimal example code that connects to the server, gets the handles and implements a basic control loop

The springs on the leg are simulated by two spherical joints and one prismatic joint set to spring-damper mode.

The servos of the robot are simply ordered to hold their initial angle and the simulation determines that the robot can indeed stand upright without any active stabilization.

6.2.2 Standing up routines

This section is heavily inspired by [?]

6.2.3 Walking

6.3 Influence on robot's design

The simulator helped shape the robot through simulations that unveiled serious design problems (inability to stand after a fall, inability to walk).

The first design is visible on fig. 6.3. It was plagued by stability problems, overcomplicated arms and simulation difficulties.

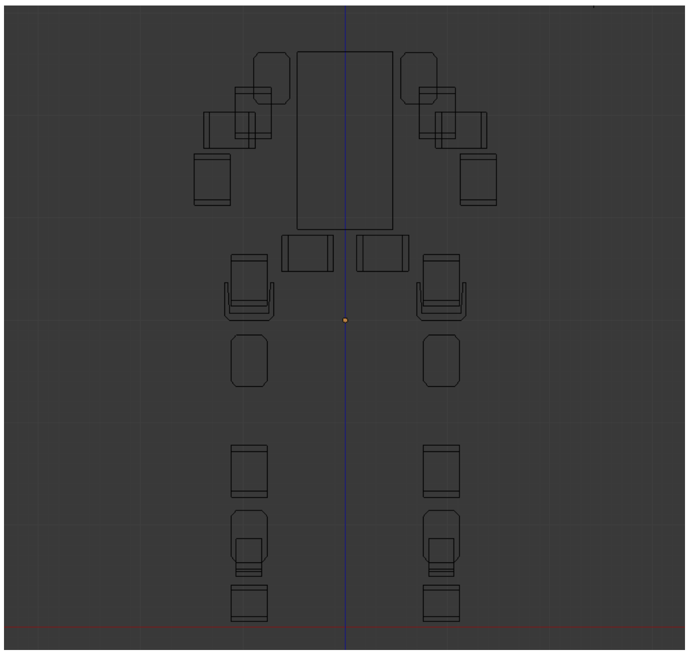


Figure 6.3: First robot design. Each arm is made of 4 servos, making them quite heavy.

The final design, visible on fig. 6.4 has better stability, wider movement possibilities and can stand up and walk more easily.

The final dimensions of the robot respect the rules of the contest:

- Height : $61.3cm$
- Height of COM : $34cm$

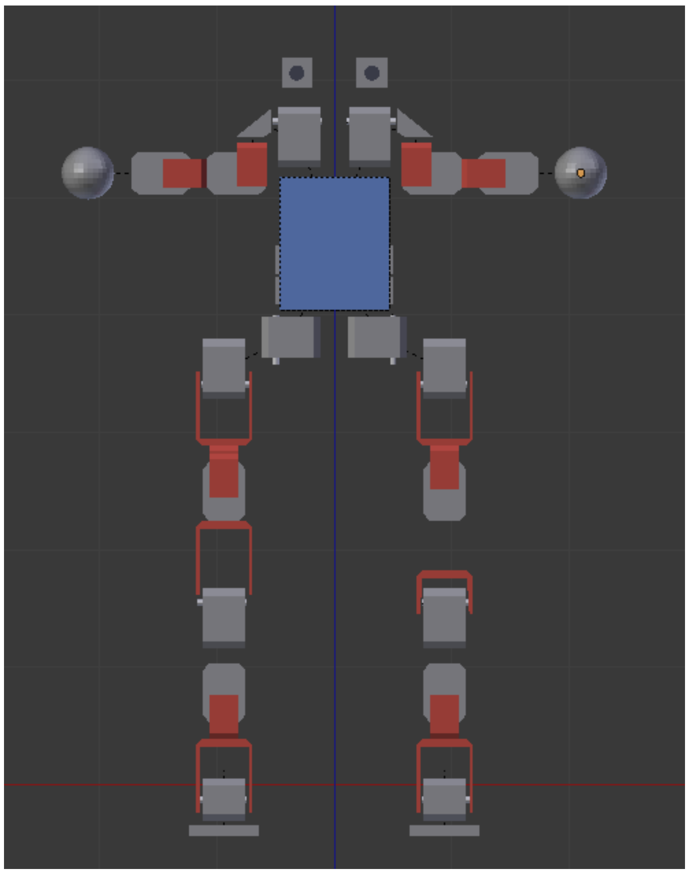


Figure 6.4: Final robot design. Arms now use 3 servos. The feet and the hips use a different configuration to have wider movement possibilities and bring down the center of gravity.

- Height of legs : cm
- Height max is $< 1.5 \times 61.3$.
- Foot area is cm^2.

Chapter 7

Conclusion

7.1 Conclusion

This is the conclusion to my work.

7.2 Future work

7.2.1 Modelling

As of now it is still uncertain if Blender shall continue to support the COLLADA format (as explained in [**?**]). In the negative, another tool should be chosen to perform the modelling.

The springs also need some work, as of now they are just there as a proof of concept but their parameters will need to be tuned.

As of now, the model uses simplified inertias, in the belief that a controller should be able to correct minor differences in behaviour between the model and the actual robot. In the case, theses inertias need to be more accurate, we suggest to use Meshlab[1] to compute the inertias of those objects.

[1]`http://meshlab.sourceforge.net/`

Appendix A

Rules

The robots that participate in the kidsize competition must respect the following characteristics[**?**]:

1. $40cm \leq H_{top} \leq 90cm$.
2. Maximum allowed weight is $20kg$.
3. Each foot must fit into a rectangle of area $(2.2 \cdot H_{com})^2/32$.
4. Considering the rectangle enclosing the convex hull of the foot, the ratio between the longest side of the rectangle and the shortest one, shall not exceed 2.5.
5. The robot must fit into a cylinder of diameter $0.55 \cdot H_{top}$.
6. The sum of the lengths of the two arms and the width of the tor so at the shoulder must be less than $1.2 \cdot H_{top}$. The length of an arm is defined as the sum of the maximum length of any link that forms part of the arm. Both arms must be the same length.
7. The robot does not possess a configuration where it is extended longer than $1.5 \cdot H_{top}$.
8. The length of the legs H_{leg}, including the feet, satisfies $0.35 \cdot H_{top} \leq H_{leg} \leq 0.7 \cdot H_{top}$.
9. The height of the head H_{head}, including the neck, satisfies $0.05 \cdot H_{top} \leq H_{head} \leq 0.25 \cdot H_{top}$. H_{head} is defined as the vertical distance from the axis of the first arm joint at the shoulder to the top of the head.
10. The leg length is measured while the robot is standing up straight. The length is measured from the first rotating joint where its axis lies in the plane parallel to the standing ground to the tip of the foot.

Appendix B

Design guidelines

For a dynamic simulation several design restrictions must be considered :

- use pure convex as much as possible, they are much more stable and faster to simulate. When a more complex shape is used, approximate it with several convex shapes.
- use reasonable sizes, neither not too small nor too big. Thin shapes may behave strangely.
- when using joints, keep the ratio of the masses below 10. Otherwise, the joint may have large orientation/position errors.

Appendix C

Control code of the servo

```lua
if not PID_P then
    PID_P=0.1
    PID_I=0
end

if init then
    pidCumulativeError=0
end
ctrl = errorValue*PID_P

if PID_I ~=0 then
    pidCumulativeError = pidCumulativeError+errorValue*dynStepSize
else
    pidCumulativeError=0
end

ctrl = ctrl + pidCumulativeError*PID_I

velocityToApply = ctrl/dynStepSize
if (velocityToApply > velUpperLimit) then
    velocityToApply = velUpperLimit
end
if (velocityToApply < -velUpperLimit) then
    velocityToApply = -velUpperLimit
end
forceOrTorqueToApply = maxForceTorque

return forceOrTorqueToApply, velocityToApply
```