University of Liège - Faculty of engineering

Master thesis

Simulation of complex actuators

Author : Hubert Woszczyk
Promotor : Prof. Bernard Boigelot

Master thesis conducted for obtaining the Master's degree in *Electrical Engineering* by Hubert Woszczyk

Academic year 2015-2016

Simulation of complex actuators

Hubert Woszczyk, under the supervision of Prof. Bernard Boigelot

Academic year 2015-2016
Faculty of Applied Sciences
Electrical Engineering

Abstract

The word *robot* has been crafted by Czech writer Karel Čapek in the beginning of the XXth century and is derived from the slavic world *robota* which means *work*, as in *there is work to be done*. Much changed since those days, and this master thesis is about robots who prefer to play football rather than work in factories. This manuscript is the result of the planned participation of a team of students to the contest *RoboCup*. A team of humanoids robots need to be build and with no prior knowledge in that subject it would be difficult to build a functioning prototype on the first try. To avoid time expensive real-life experiments, a simulation tool is needed. We will take a survey of the existing simulators and choose the one that fits our needs best before using it to perform some basic simulations on the model of our robot. These simulations are used to detect design flaws that made the robot unable to stand up from a prone position or unable to walk. The end result of this work is the design of a robot that is able to stand and stand up when toppled over. Furthermore, this report serves as confirmation of various design decisions that were made beforehand such as the choice of the MX-28R as the servo to be used in the joints.

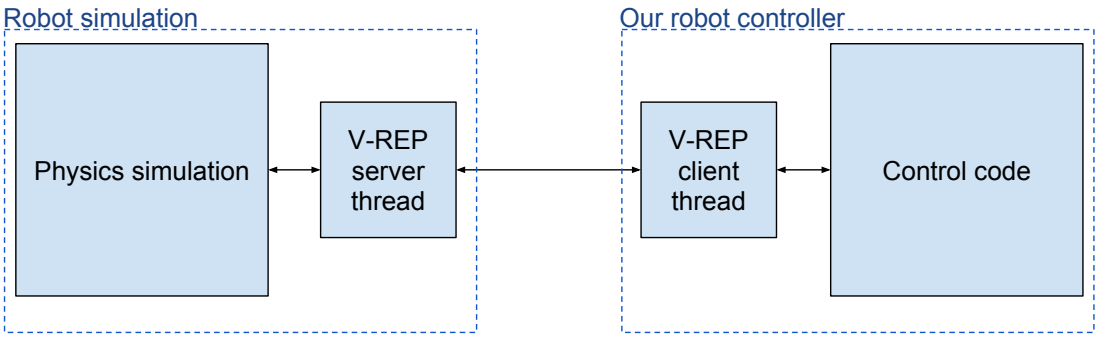
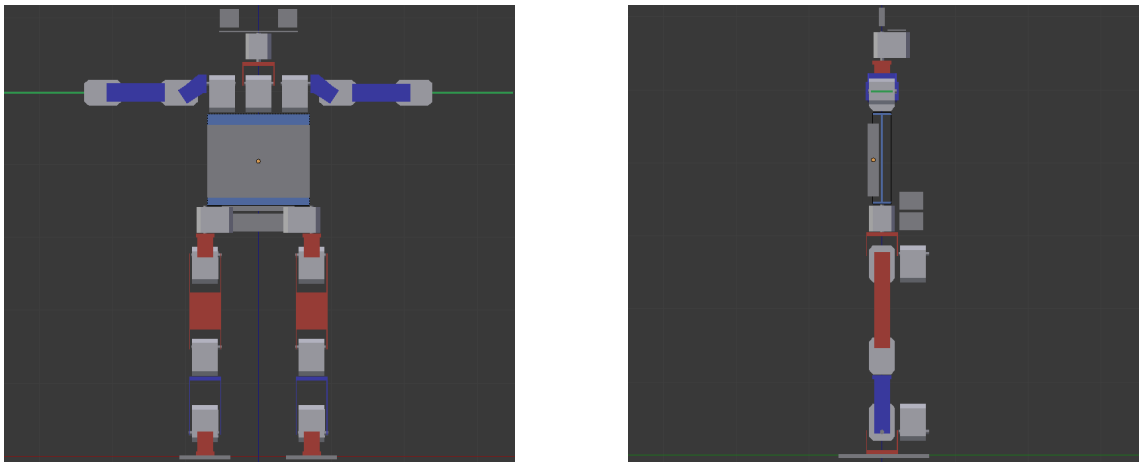


Figure 1: Representation of the architecture of the simulation setup. While the physics simulation is done in a simulator (V-Rep) along with the local control of the servos, the higher level control algorithms are executed outside. The communication between the simulator and high level control code is based on a TCP socket.



(a) Front view of the robot

(b) Side view of the robot

Figure 2: Front and side views of the final robot design this master thesis produced. The servos (in grey) are connected together through different types of frames (in red and blue). At the top, two cameras are discernible and at the centre, the electronics and batteries can be seen.

Acknowledgements

My first thanks go to Prof. Bernard Boigelot who made it possible for numerous students, including me, to work in the passionate field of robotics. I also wish to thank him for his guidance, help and accessibility.

I am deeply grateful to my friends Elodie and Laurine for reading and correcting this manuscript. I also want to thank fellow students Grégory Di Carlo and Guillaume Lempereur with whom I had the pleasure of working together one last time.

Finally, I would like to express my sincere thanks to all those who helped me complete this master thesis.

Contents

List of Figures

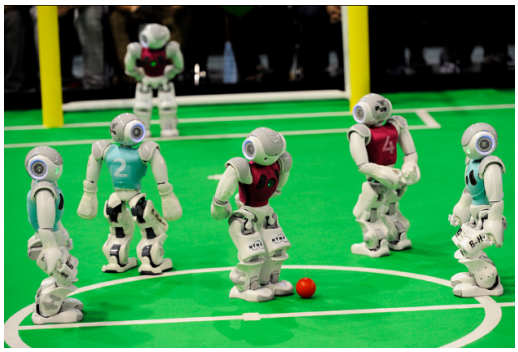
List of Tables

Chapter 1

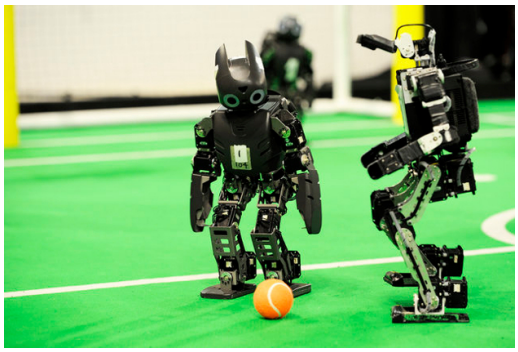
Introduction

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's 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 chapter 3 that explain the problem we want to solve and motivates the choice of V-Rep as the main simulation tool for this project.

Chapter 4 is about the modelling of the building blocks of our humanoid robot in preparation for chapter 5 to go into the core of the subject with some simulations. This chapter also explains how our work influenced the design of the robot.

Chapter 6 concludes this work by summing up what is achieved and laying out future prospects.

Chapter 2

Principles of interactive rigid body simulation

This chapter briefly introduces the basic concepts relative to physics engines.

2.1 Problem statement

Physics engine trouble themselves with the simulation of classical Newtonian 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. In our case, 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.

2.2 Overview of the components of a simulator

The section is heavily inspired by Jan Bender, Kenny Erleben, Jeff Trinkle and Erwin Coumans' work in [BETC12]. Bender *et al.* [BETC12] tells us that the simulation of rigid body dynamics is usually built around the loop presented in fig. 2.1.

The simulator begins by finding the collision points between objects (see section 2.3). These points are then used to derive motion laws which are solved to determine the forces that act on the objects and prevent them from inter-penetrating (see section 2.4). 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.

2.3 Collision detection

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

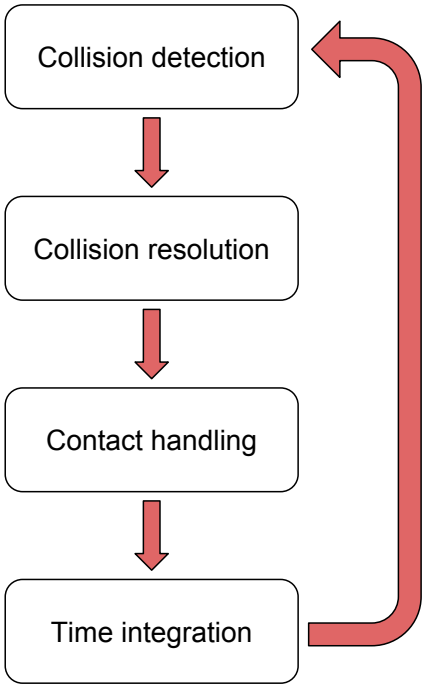


Figure 2.1: Modular phase description of the sub tasks of a rigid body simulator.

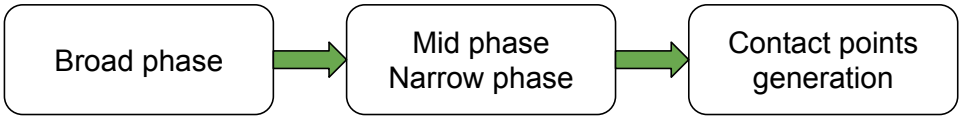


Figure 2.2: Modular description of the collision detection in a physics engine. The mid phase and narrow phase are grouped together because they are often combined for performance reasons.

During the broad phase, objects are approximated by simple geometric primitives as 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 used in the collision resolution part of the simulation loop.

We won't discuss collision detection further. For our foreseen application it is sufficient to know that simpler shapes are easier to handle. For further discussion, different possible implementations are detailed in [JTT01].

2.4 Collision resolution

When bodies collide, high forces of very short duration are exerted. In the case of rigid bodies the duration is infinitesimal and the forces become infinite. This is problematic because we can see from eq. (2.1) that infinite forces yield infinite accelerations which makes direct integration of that equation impossible and breaks the simulation loop.

A solution to this was proposed by Mirtich in [Mir96]. The idea is to use the standard integration rule of the simulator up to the collision, use an impulse-momentum based collision rule to determine the velocities after impact and then resume the integration.

Several collision rules exist. Newton's Hypothesis states that

$$\mathbf{v}_n^+ = e\mathbf{v}_n^-$$

where $\mathbf{v}_n^+$ is the relative normal velocity after impact, $\mathbf{v}_n^-$ is the relative normal velocity before impact and $e \in [0, 1]$ is called the coefficient of restitution. When $e = 1$ the collision is perfectly elastic and when $e = 0$ all the energy is lost.

2.5 Formulating the nonlinear complementarity problem (physics model)

The model is based on the three laws of Newton :

- The velocity of an object remains unchanged if no force act upon it.
- The rate of change of the momentum of an object is equal to the force acting on it.
- For every force there is an equal and opposite force.

2.5.1 Kinematics

The position of an object in a 3D space is given by a vector $\mathbf{p} \in R^3$ 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 is represented by either Euler angles or unit quaternions (Q_s, Q_x, Q_y, Q_z)

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). While Euler angles are easier to interpret, they have the disadvantage of the gimbal lock, so quaternions are usually preferred. 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.

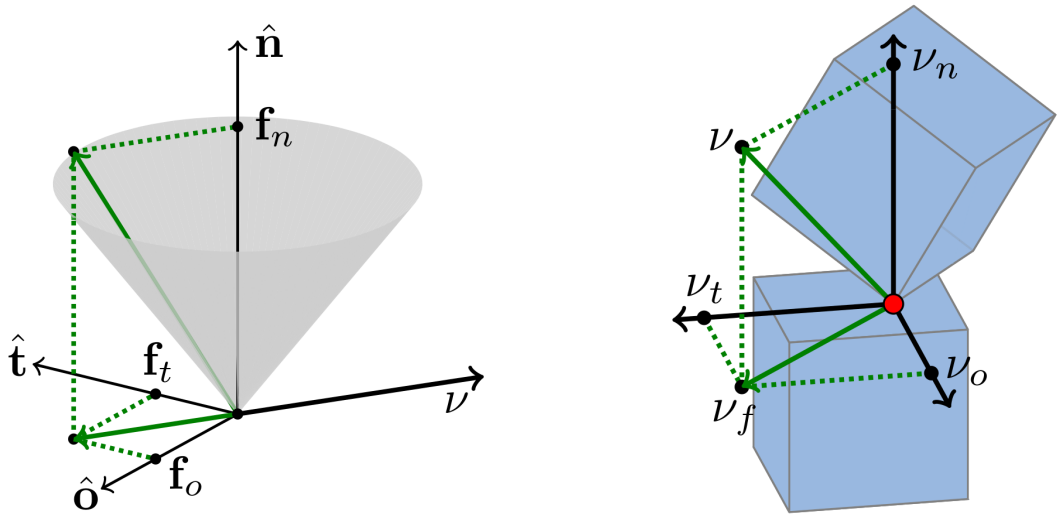
2.5.2 Newton-Euler

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

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

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

2.5.3 Friction



(a) A friction cone

(b) Contact velocities

Figure 2.3: Friction cone of a contact and decomposition of the contact force and relative velocity

Friction is the most complex type of force a physics engine must deal with.

Chapter 3

Choice of the simulation platform

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 in a 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 servo used in the robot will have several sensors such as accelerometers, rotary position encoders. We want to model them in the simulator since they will be used by the control code.

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 [ETT15].

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 a 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 but binaries are not provided for Windows and 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. Furthermore, the fact that the scripting language is Python make code development faster and the latter's support of TCP sockets 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 the development roadmap [Ble15].

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 :

– 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. Moreover, it supports TCP sockets 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 :

– 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

We make the choice of using a simulator rather than just a physics engine for several reasons :

- The product of this master's thesis is expected to be used by another students next year and they would rather spend as little time as possible learning how to use it. Hence an established simulator rather than an in-house one is preferred.
- A simulator eliminates the need of developing 3D visualization by ourselves.
- A simulator already provides much needed features as the import of 3D models or an interface to the settings of the simulations.
- Most simulators have an already defined API for remote control.

From the simulators we surveyed earlier in this chapter the best overall seems to be V-Rep.

This choice is further confirmed by Ivaldi in [IPPN14] who shows that V-Rep is amongst the highest rated tools amongst roboticists.

3.5.2 Physics engine

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 [HWS+12] where Newton Dynamics is stated to be the best engine when it comes to handling a high number of constraints.

3.5.3 Modelling software

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

[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 focuses on the modelling of the atomic elements of the robot. We begin by explaining how to create the model of an object inside V-Rep. We then show how to use to model simple elements such as frames or electronics who are not active during the simulation i.e they do not perform any function. Those can be represented strictly mechanically. We finally show how to model active pieces such as the cameras of the servos.

4.1 Problem statement

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

- **Miscellaneous mechanics and electronics:** A type of element that must be included in the model are the frames, Figure 4.1b shows the FR07-H101 frame. On the actual robot, they will link the servos together. Other mechanical elements are the hands, the feet and the plate that will act as the trunk of the robot.

  In addition to all the aforementioned elements we plan to equip the legs of the robot with springs. Their purpose will be to

  The robot will also have some an array of electronic elements. A motherboard (Figure 4.1d) to give orders to the servos, batteries to stock the energy necessary to power them and the electronics to convey that energy through the robot.

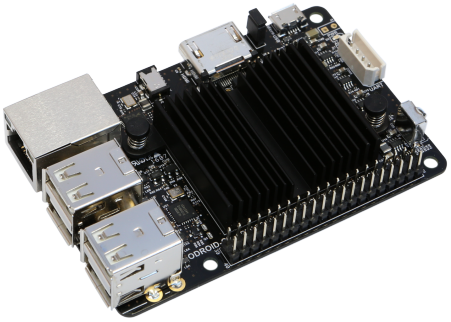
- **Cameras:** The robot will be equipped with two cameras of the type shown in Figure 4.1c. They will be used by the robot to locate itself in the play arena. As we may test some machine vision algorithms during our (future) simulations we need to have the ability of retrieving the image a camera captures.

- **Servos:** We will be using the MX-28R servo, shown in Figure 4.1a, as the driving element of the joints of the robot. We need to represent it as a rigid body as well as a mechanical device that can generate torque on its axle and hold a position that is ordered to hold.

(a) MX-28R servo.

(b) FR07-H101 hinge type frame.

(c) LI-USB30-M021C camera.



(d) Odroid C-2 motherboard.

Figure 4.1: Atomic elements of our robot. The MX-28R and the camera are more complex to model as they each perform a function in the simulation. On the other hand, the frame and the motherboard are only dead weight.

4.2 Modelling in V-Rep

The creation of a model inside V-Rep starts with the creation of a mesh[1] in a modelling tool, Blender in our case. It is preferred to have a convex mesh, i.e one that whose all interior meshes are less or equal to 180°, as they are easier for the simulator to handle. When it is absolutely necessary to keep the mesh concave, a solution is to separate it into several convex meshes that will be grouped together once in V-Rep. Once saved in a file format recognized by V-Rep the mesh can be imported into the latter.

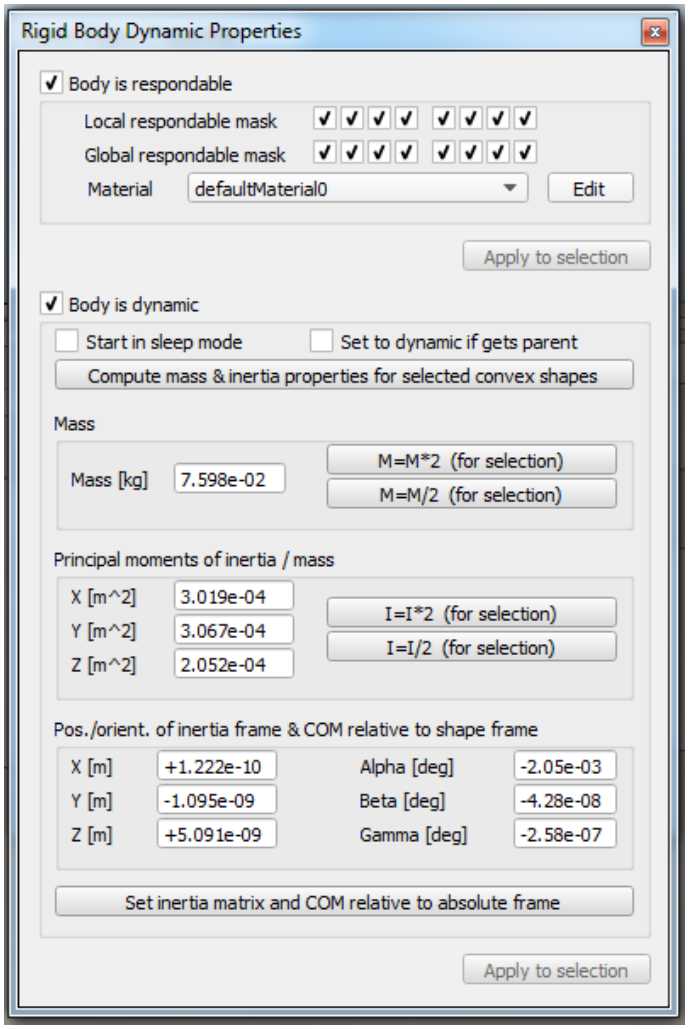


Figure 4.2: Configuration window of the dynamic properties of the selected rigid body. We can make it *respondable*, i.e it should collide with other objects. We can also make it *dynamic* in which case it will be dynamically active in the simulation and react to forces. When dynamically active, the movement of this body will be influenced by its *mass* and its *inertia*. It is also possible to set what is called *Material* by V-Rep.

When in V-Rep all that is left is marking the body as dynamic, respondable (if necessary) and setting the mass and the inertia. It is possible to have V-Rep compute it automatically from the geometry and the density but it can also be set manually.

[1] an ensemble of vertices and faces that represent an object

4.3 Modelling the miscellaneous mechanics and electronics

The modelling of most of the mechanical elements and electronics is rather straightforward but there are some where the process is more troublesome. This is the case of the feet, frames and springs and those will be detailed hereunder after a short explanation of the modelling process of the remaining pieces.

4.3.1 Modelling the electronics, batteries, hands and the central plate

The modelling of these elements consists in following the procedure explained in Section 4.2. All the necessary informations are contained in Table 4.1.

Module	**Weight** $[g]$	**Material**	**Density** $[kg/m^3]$	**Dimensions** $x[mm] \cdot y[mm] \cdot z[mm]$
Odroid C-2	40	[-]	840	$85.0 \cdot 56.0 \cdot 10.0$
Li-Po battery	188	Li-Po	2304	$103.0 \cdot 33.0 \cdot 24.0$
Hand	30	Aluminium	3000	$70.0 \cdot 31.0 \cdot 31.0$
Feet	52	Polymer	1000	$70.0 \cdot 125.0 \cdot 6.0$
Central plate	209	Aluminium	4000	$140.0 \cdot 30.0 \cdot 124.5$

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. The differences in density between different pieces made of the same material represent the differences in geometry, i.e pieces with holes in them that are not modelled but taken into account by making the piece lighter.

4.3.2 Modelling the feet

In essence, the feet of our robot will be plates that will be fixed to the last servos of the legs. Contrary to other elements, feet *must* be respondable as they come into contact with the floor. It follows that the friction parameter will be important too and it will have to be chosen in accordance to the friction parameter of the material they will be made of.

It should be noted that for the needs of the simulation, their z dimension has been exaggerated up to $6mm$ because of collision problems (sometimes, the contact point was on the upper side of the feet while it should have been on the down side) that disturbed the course of the simulation. Therefore, the z dimension reported in Table 4.1 is the double of the actual one and the density has been halved to keep the same mass.

4.3.3 Frames

Frames are a special case: in the actual robot they are essential in connecting the servos together but they are not necessary in the simulation. Indeed the link between servos can be made directly through the joints without the intermediary of a frame.

This reduces the accuracy of the model but not by much and we gain in speed and stability by reducing the number of constraints and not having to integrate a convex shape into the model.

4.3.4 Springs

By itself a spring can be modelled by a prismatic joint linking two dynamic objects together. The prismatic joint is then set to *spring-damper* mode and the stiffness and damping can be specified.

However, if we want to add it to a leg we have to somehow create a loop in a top down hierarchy. Moreover, we need to take into account displacements of the anchors.

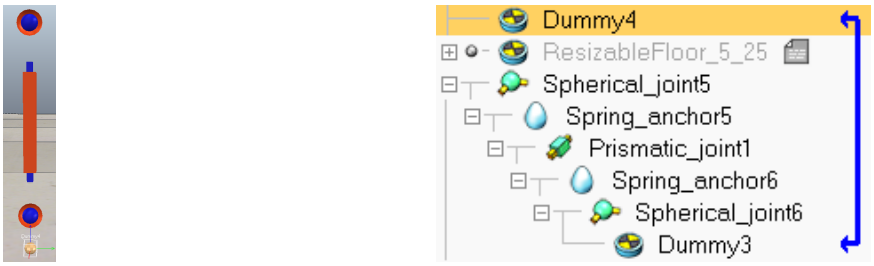


Figure 4.3: How to model a spring inside V-Rep.

4.4 Modelling the cameras

The robot will be equipped with two small cameras, of the type illustrated in **??**, used to locate the robot on the field[2].

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 :

- 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.

[2]This is the subject of another master's thesis this year.

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
Density	2213	kg/m^3
Inertia		gmm^3
Resolution		px^2
Field of view		$^\circ$

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

4.5 Modelling the MX-28R

This section explains how the MX-28R is characterized and modelled.

4.5.1 Determining the continuous torque

The primary parameter we must know the value of to properly model the MX-28R is the torque it generates. The manual ([Dyn16a]) provides the value of the stall torque ($2.5Nm$ @$12V$) and a graph showing the efficiency of the servo at different torques and speeds but it does not provide a value of the actual continuous torque. We thus compute it from the maximal torque of the DC motor (référence à mettre !) and the reduction ratio of the gears.

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

The continuous torque is thus equal to $0.7Nm$.

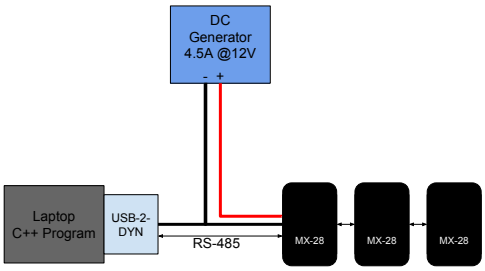
4.5.2 Determining the stall torque

The stall torque is determined through a small experiment with a real MX-28R. The experimental setup is explained in Figure 4.4 and Figure 4.4b. The experiment itself is presented in Figure 4.5.

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 $1.64Nm$. The complete results are listed in Table 4.3.

4.5.3 Determining the rotation speed

In this experiment we will test some simple dynamics. The setup is shown in fig. 4.6. The goal is to tune the P parameter of the PID controller inside V-Rep. In order



(a) The MX-28 servos are powered by a DC generator and controlled by a laptop equipped with a USB2DYNAMIXEL(USB-2-DYN) device.

(b) A USB2DYNAMIXEL. It turns an USB port into a serial port (RS485, TTL or classic serial connector) that can be used to control Dynamixel manufactured servos. [Photo from [Dyn16b]]

Figure 4.4: Experimental setup

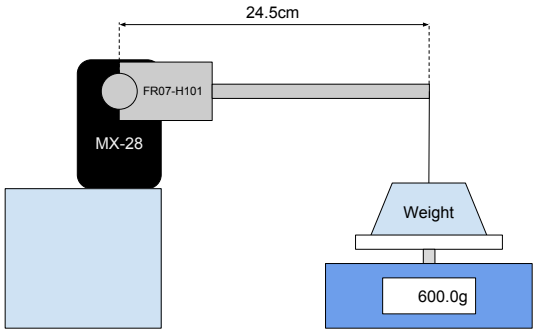


Figure 4.5: Experimental setup for torque testing. A weight w of is suspended at a distance d from the servo, resulting in a applied torque of $w \times g \times d$. The goal is to find the weight w for which the servo is unable to lift the arm.

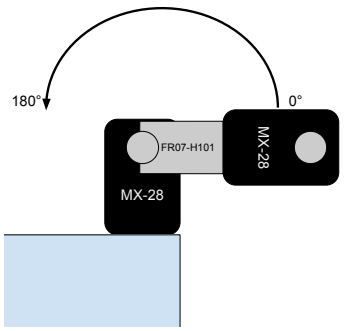


Figure 4.6: Experimental setup for dynamics testing. One servo lifts the other. The goal is the measure the time it takes to swing the arm from 0° to 180°.

to achieve this, we film the real servo going from 0° to 180° and measure the time it takes.

Executing this manoeuvre at $12V$ with no speed limit imposed takes $620msec$.

We then reproduce the same manoeuvre inside V-Rep, measuring the time through code. We modify the proportional gain of the PID controller until we get the same time.

4.5.4 Results

	Data	**Unit**
Weight	77	g
Dimensions	$35.6 \times 50.6 \times 35.5$	mm^3
Ixx	$22,649$	$g \cdot mm^4$
Iyy	$12,868$	$g \cdot mm^4$
Izz	$17,733$	$g \cdot mm^4$
Announced stall torque @11.1V	2.1	Nm
Experimental stall torque @11.1V	1	Nm
Announced stall torque @12V	2.5	Nm
Experimental stall torque @14.8V	1.2	Nm
Announced stall torque @14.8V	3.1	Nm
Experimental stall torque @12V	1.6	Nm
Nominal continuous torque @12V	0.7	Nm
Announced unloaded rotation speed @12V	55.00	Rpm
Experimental unloaded rotation speed @12V	48.33	Rpm

Table 4.3: Characteristics of a MX-28R type servo. Data taken from [Dyn16a] and from http://www.robotis.com/view/DXL-INERTIA/RX-28_INERTIA.pdf[Accessed 4/6/2016].

4.5.5 Modelling the MX-28R

(a) MX-28R servo. **(b)** Model of the MX-28R.

Figure 4.7: 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.

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 1.2, the maximum torque developed my Mx-28 servos as shown by our earlier experiments (**??**).

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

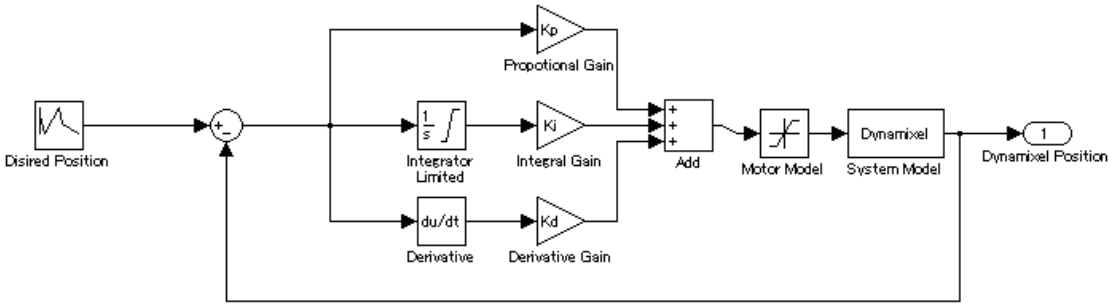


Figure 4.8: MX-28 PID controller, as represented in [Dyn16a]. It is a standard PID controller although by default only the P parameter is non-zero. The function of the *motor model* block is to clamp the computed correction torque in the allowed interval whereas the *system model* block is a simulink representation of the MX-28R and does not exist in the actual implementation inside the latter.

Chapter 5

Applications

This chapter explains how we can use the simulator and the pieces we modelled in the preceding chapter to model a whole robot. We then showcase some simulations that were used to validate robot designs and finally present the influence these simulations had on the final design of the robot.

5.1 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. 5.1.

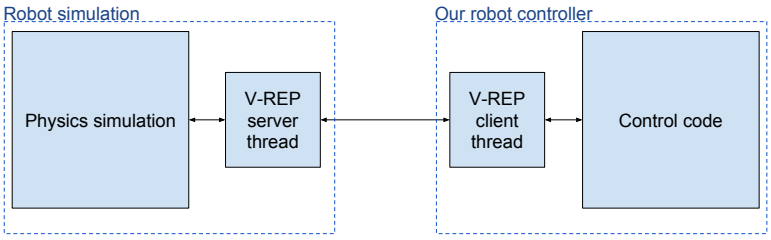


Figure 5.1: V-Rep simulates the robot while an external program sends order to the robot over TCP/IP thanks to the client/server thread provided by V-Rep.

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 5.2 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.

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

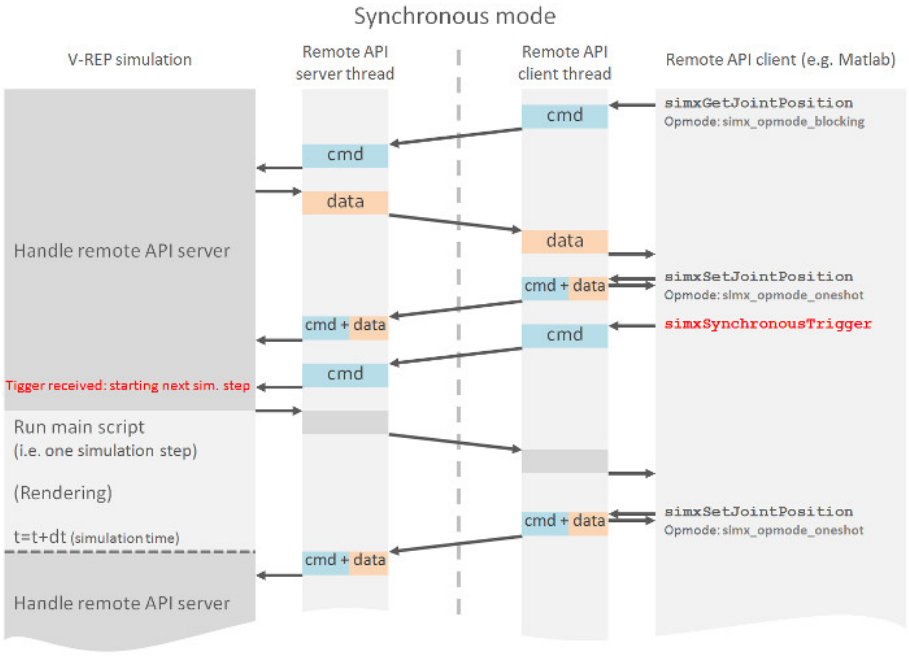


Figure 5.2: Typical interaction between the simulator and the control code. The simulation runs on two threads : the simulation and the server thread. The server threads can receive orders from a client thread which is controlled by a custom application of our own. The simulator waits for a trigger before simulating the next timestep.[Rob16]

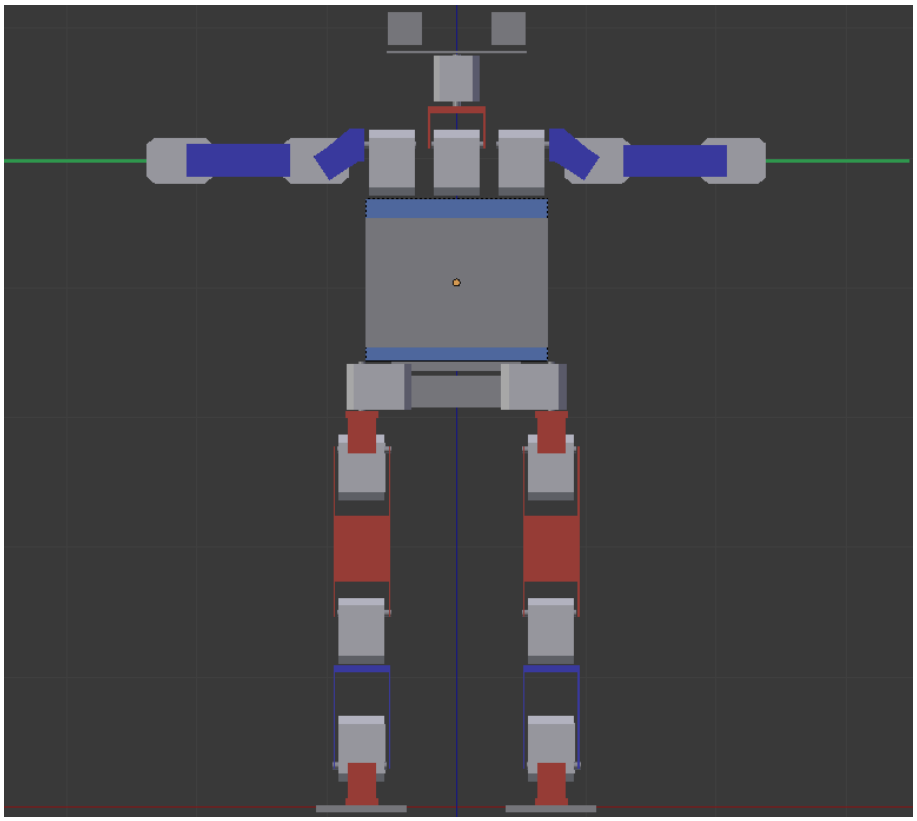
- simxSetJointTargetPosition : this function sets a target position for a joint.
- simxSetFloatSignal : this function gives the possibility of setting the value of a signal inside V-Rep. A signal is a variable created during the simulation that is accessible both to the simulator and the external world. This is useful if we want to extend the interface that V-Rep provides.
- simxGetFloatSignal : this function retrieves the value of a float signal.

5.2 Applications

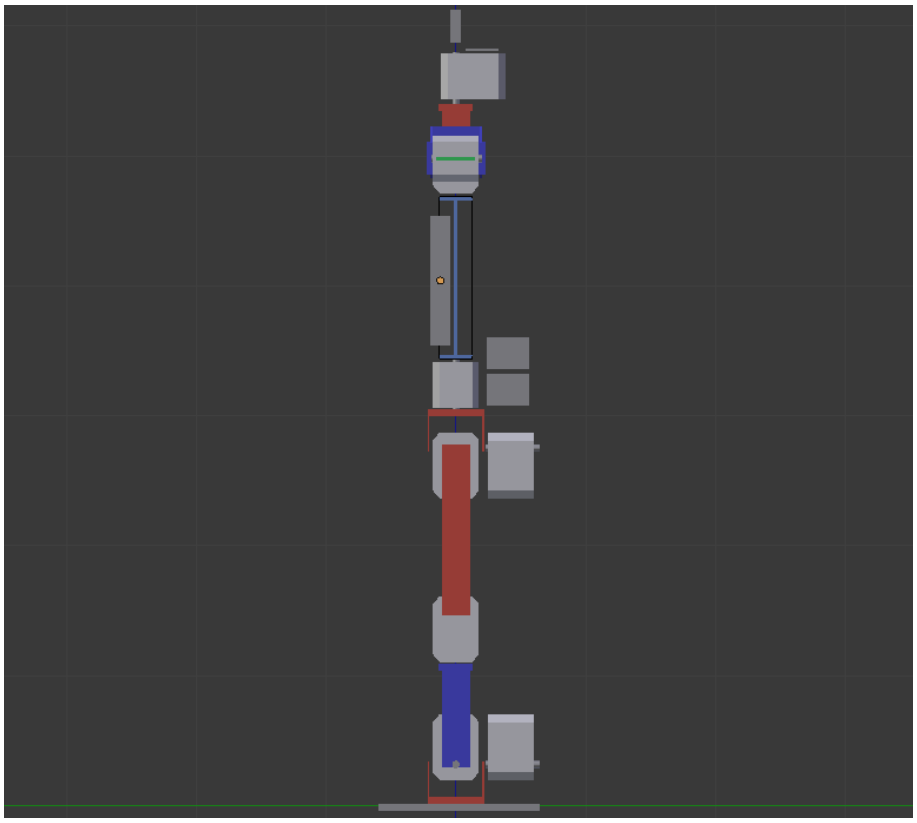
5.2.1 Static stability

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. That model is presented in fig. 5.3

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. This is a simple test that can rule out bad designs quite easily.



(a) Front view of the robot



(b) Side view of the robot

Figure 5.3: Front and side views of the final robot used in the simulations presented in this chapter. The grey servos are connected together through different types of frames. At the top, two cameras are discernible and at the centre, the electronics and batteries can be seen

5.2.2 Going from a supine to a prone lying position

The main motivation for a routine that makes the robot move from a supine[2] lying position to a prone[3] one is that it allows us to only have one standing up routine.

The routine is defined as follows :

1. From 0 to 0.39*sec* : the robot brings his right arm above his head while preparing the left one to lift its body from the left side. Both legs are twisted towards the right side at the hips.
2. From 0.40 to 0.99*sec* : the left leg swings towards the right side while the right leg swings towards the left side. The left elbow prepares to push.
3. From 1.00 to 1.59*sec* : the left elbow pushes the body up and the hips untwist. The left feet touches the ground on the right side.
4. At 1.6*sec* : The robot relaxes all its limbs and is now prone.

The evolution of the angles held by the joints during the routine is presented in fig. 5.4 and fig. 5.5.

5.2.3 Standing up from a prone lying position

This routine was inspired by Jörg Stückler, Johannes Schwenk, and Sven Behnke in [SSB06]. It is defined as follows:

1. From 0.00 to 0.19*sec* : in preparation for the arms to lift the body in the next step we adjust the roll of the shoulders.
2. From 0.20 to 0.49*sec* : in order to reduce the stress on the servos of the arm, we bend the arms a little before the next step.
3. From 0.50 to 0.89*sec* : lift the body up through the hips with the help of the arms. This step and the next are preparation for the next step where we will move the feet underneath the body.
4. From 0.90 to 1.39*sec* : now that the body is held up by the arms we reverse the holding angle for the hips, to move the hips up.
5. From 1.40 to 1.99*sec* : we move the feet underneath the body by bending the knees and bringing the legs closer to the body. The objective is to bring the center of mass inside the support area of the feet.
6. From 2.80 to 4.00*sec* : the robot slowly gets up by unbending the knees and the hips.

[2] lying on the back
[3] lying on the belly

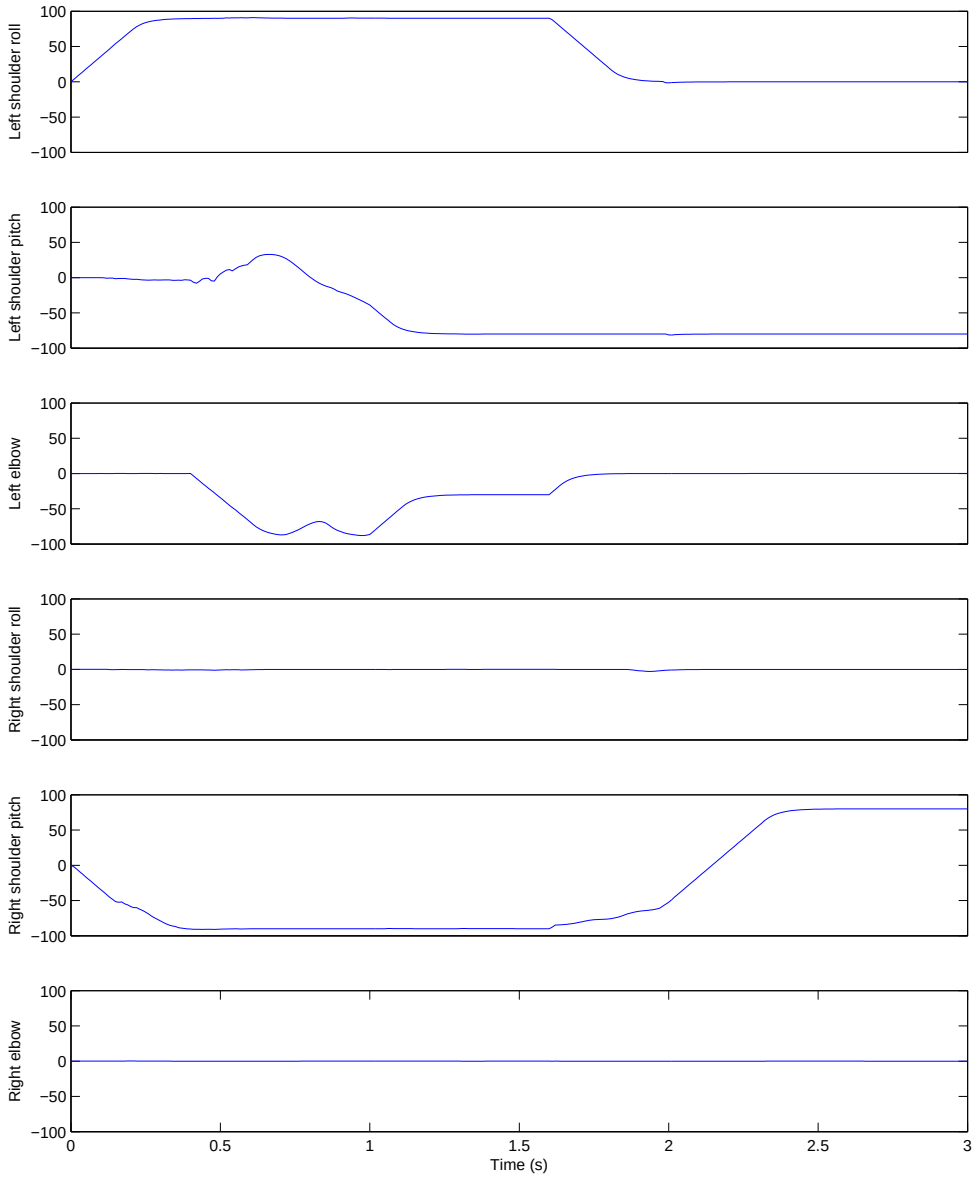


Figure 5.4: Angles of the arms during *supine* to prone manoeuvre. Predictably it is the left arm that is the most active as it is the one that is used to make the robot roll over. We notice that at times (around $0.7sec$) the left elbow does not generate enough torque to hold the desired angle.

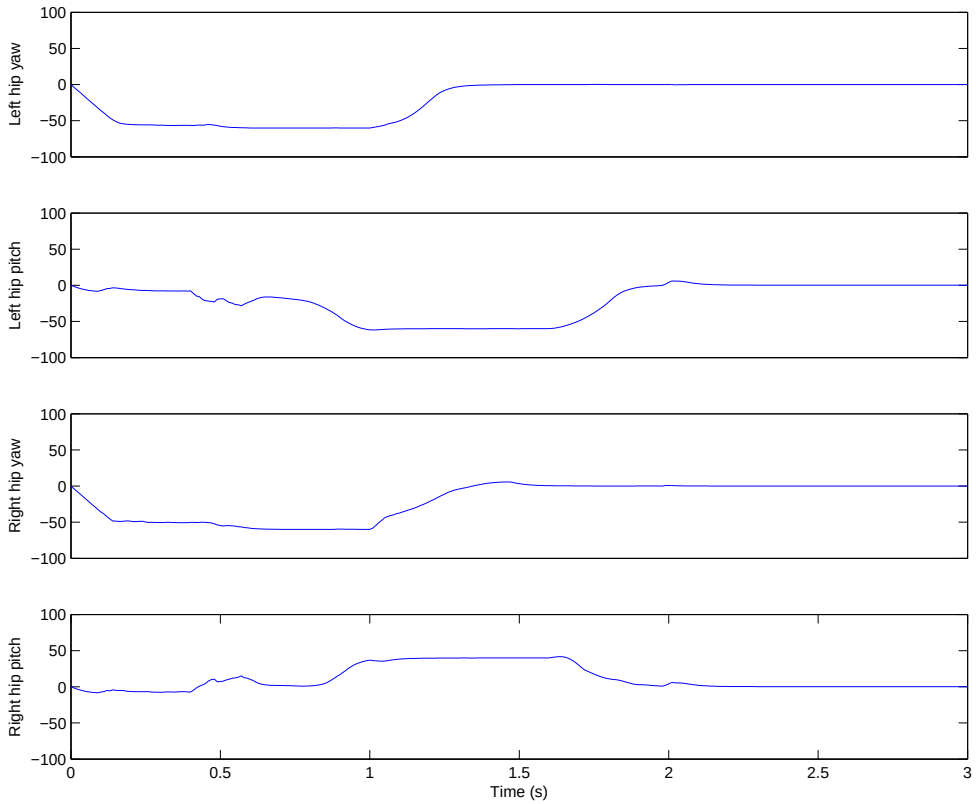


Figure 5.5: Angles of the legs during *supine to prone* manoeuvre

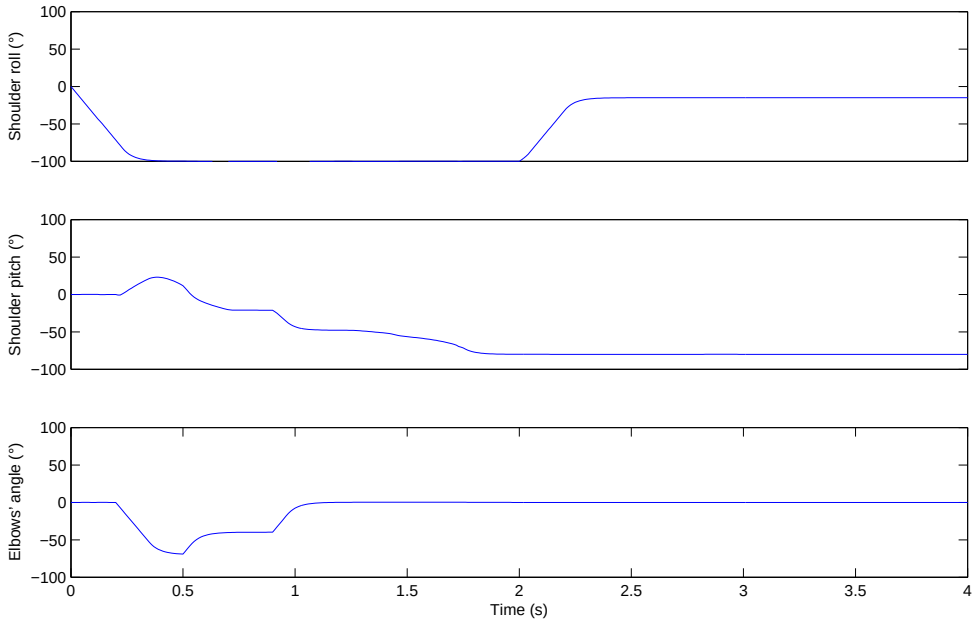


Figure 5.6: Angles of the arms during *supine* to prone manoeuvre

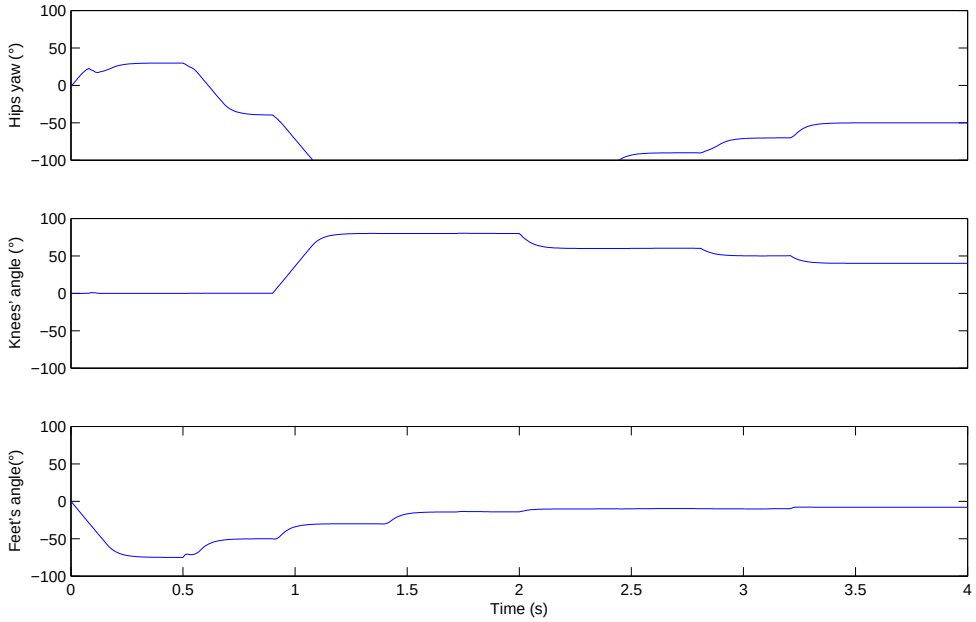


Figure 5.7: Angles of the legs during *supine to prone* manoeuvre

5.3 Influence of the simulations on the final design of the robot

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. 5.8. It was plagued by stability problems, overcomplicated arms and simulation difficulties.

The final design, visible on fig. 5.9 has better stability, wider movement possibilities and can stand up and walk more easily.

The final dimensions of the robot abide by the rules of the contest (see appendix A) :

1. Height (H_{top}) : $40 \leq 61.75 \leq 90cm$.
2. Weight : $2.726kg$
3. Height of COM (H_{COM}) : $34cm$. Foot area : $175 < cm^2$.
4. Foot aspect ratio : $1.79 \leq 2.5$.
5. Minimal width of the robot : $20 \leq 33.96cm$
6. $69.72 \leq 74.10cm$
7. Maximal height : $75.52 < 92.63cm$.
8. Leg length (H_{leg}) : $21.61 \leq 27.5 \leq 43.23cm$
9. Height of the head (H_{Head}) : $3.09 \leq 10.75 \leq 15.44cm$

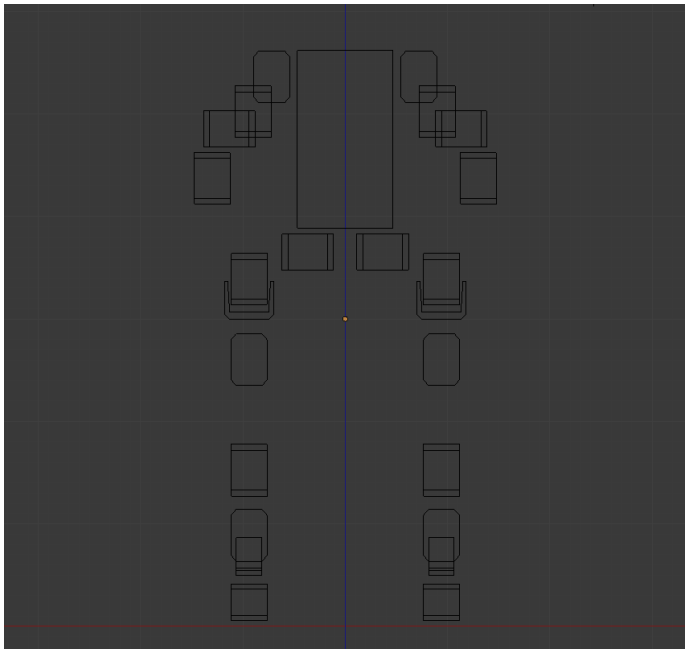


Figure 5.8: First robot design. Each arm is made of 4 servos, making them quite heavy.

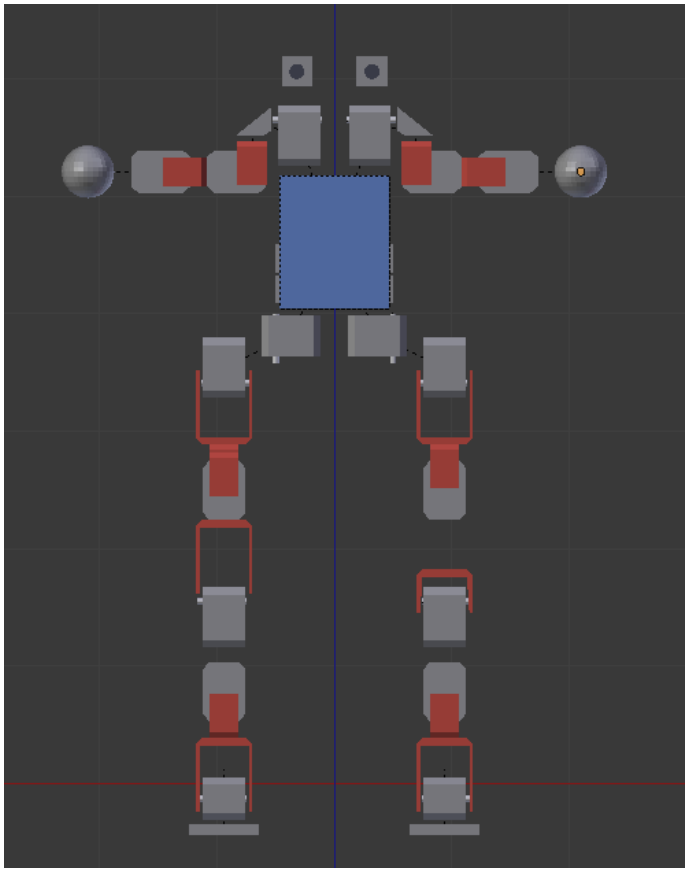


Figure 5.9: Final robot design. Arms now use 3 servos. The feet and the hips use a different configuration to have wider movement possibilities and bring down the center of gravity.

Chapter 6

Conclusion

6.1 Conclusion

In this work we applied rigid body simulation to the problem of designing a robust humanoid robot. Our work produced a model of a robot that respects the rules of the kidsize league of the RoboCup Soccer competition. Our simulations prove that it is able to stand up from a lying position, be it prone or supine.

6.2 Problems encountered

A master's thesis is a major endeavour and these are rarely devoid of obstacles. During this year several elements obstructed the completion of this work :

- V-Rep is a fine tool but the lack of a proper internal modelling tool was a major thorn in the side as every major modification meant that the whole model had to be modified in Blender and re-imported into V-Rep. This created a lot of overhead work which contributed nothing of interest to this work.

  This drawback is generalized amongst all the simulators that we surveyed at the beginning of this report and we feel it should be addressed quickly by their creators. Nevertheless, we understand that a modelling tool such as Blender took years to create so we would not expect simulators to catch up any time soon.

- We also learned of the importance of studying mechanisms carefully before using them. Though things might appear simple at first, subtle implementation details might change everything. A fine example of that is us melting the core of the motor inside a MX-28R servo during our tests because of our trust in the announced safety mechanisms. Needless to say, they proved insufficient and we should have examined the documentation more closely.

- Choosing a physics engine was difficult because the field is quite fragmented. On one hand there exist well established commercial solutions, but they are focused on games and make some significant shortcuts whenever possible in order to be as fast as possible. On the other hand there exist a quantity of open-source physics engine but they are usually the work of one man and are

poorly documented. It was hard to motivate the choice of Newton Dynamics on any other basis than 'it worked best'.

6.3 Future work

6.3.1 Modelling

While the model is in a usable state it could still be bettered and we suggest to begin with the items listed hereafter:

- **Springs.** 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.
- **Inertias.** 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. If these inertias need to be made more accurate, we suggest to use Meshlab[1] to compute the inertias of those objects.
- **Model format.** It is still uncertain if Blender shall continue to support the COLLADA format, as explained mentioned in the development roadmap ([Ble15]). In the negative the choice should be made whether to continue using the COLLADA format and find another modelling software that supports it or to move on to another format (URDF for example, now that the robot model is defined).

6.3.2 Routines

Now that we have a simulator and a complete model of the robot, more routines can be created.

- **Standing up from a supine position.** Even though the robot can roll from a supine to a prone lying position and use the standing from prone routine it would be faster to be able to stand from a supine position directly.
- **Walking.** Being able to walk is the basic requirement for a robot to compete in RoboCup. A walking sequence is the last proof needed to be able to tell that the robot we designed is able to compete.
- **Shooting a ball.** As soon as the robot is able to walk, the next step should be testing if it can shoot a soccer ball.

6.3.3 Online simulation

In parallel or after creating the routines aforementioned, the simulator should be used to test the high level control code of the robot. The interaction between the simulator and the control code will be the same but the control code will be much more complex than just a static sequence of orders.

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

Bibliography

[BETC12] Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. Interactive simulation of rigid body dynamics in computer graphics. In *EUROGRAPHICS 2012 State of the Art Reports*, pages 95–134. Eurographics Association, 2012.

[Ble15] Blender. 2.8 project developer kickoff meeting notes, 2015. https://code.blender.org/2015/11/the-2-8-project-for-developers/ [Accessed 01-May-2016].

[Dyn16a] Dynamixel. Robotis e-manual v1.27.00, 2016. http://support.robotis.com/en/product/dynamixel/mx_series/mx-28.htm [Accessed 18-April-2016].

[Dyn16b] Dynamixel. Robotis e-manual v1.27.00, 2016. http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm [Accessed 18-April-2016].

[ETT15] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. *International Conference on Robotics and Automation*, 2015.

[HWS+12] Johannes Hummel, Robin Wolff, Tobias Stein, Andreas Gerndt, and Torsten Kuhlen. An evaluation of open source physics engines for use in virtual reality assembly simulations. In *Advances in visual computing*, pages 346–357. Springer, 2012.

[IPPN14] Serena Ivaldi, Jan Peters, Vincent Padois, and Francesco Nori. Tools for simulating humanoid robot dynamics: a survey based on user feedback. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 842–849. IEEE, 2014.

[JTT01] Pablo Jiménez, Federico Thomas, and Carme Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[Mir96] Brian Vincent Mirtich. *Impulse-based dynamic simulation of rigid body systems*. PhD thesis, University of California at Berkeley, 1996.

[Rob15] Robocup. Robocup soccer humanoid league rules and setup, 2015.

[Rob16] Coppelia Robotics. V-rep user manual 3.3.0, 2016. http://www.coppeliarobotics.com/helpFiles/index.html [Accessed 18-April-2016].

[SSB06] Jörg Stückler, Johannes Schwenk, and Sven Behnke. Getting back on two feet: Reliable standing-up routines for a humanoid robot. In *IAS*, pages 676–685, 2006.

Appendix A

Rules

The robots that participate in the kidsize competition must respect the following characteristics[Rob15]:

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 torso 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

```
1  if not PID_P then
2      PID_P=0.1
3      PID_I=0
4  end
5
6  if init then
7      pidCumulativeError=0
8  end
9  ctrl = errorValue*PID_P
10
11 if PID_I ~=0 then
12     pidCumulativeError = pidCumulativeError+errorValue*
           dynStepSize
13 else
14     pidCumulativeError=0
15 end
16
17 ctrl = ctrl + pidCumulativeError*PID_I
18
19 velocityToApply = ctrl/dynStepSize
20 if (velocityToApply > velUpperLimit) then
21     velocityToApply = velUpperLimit
22 end
23 if (velocityToApply < -velUpperLimit) then
24     velocityToApply = -velUpperLimit
25 end
26 forceOrTorqueToApply = maxForceTorque
27
28 return forceOrTorqueToApply, velocityToApply
```