Manufacturing, Control and Testing of a Tensegrity Robot for Planetary Landing and Exploration

Master Project

Marc André Leroy

Microengineering Section

Biorobotics Laboratory
Prof. Auke Ijspeert
Assistant: Florin Dzeladini
School of Engineering
École Polytechnique Fédérale de Lausanne

Dynamic Tensegrity Robotics Lab
Vytas SunSpiral
Supervisors: Massimo Vespignani and Jonathan Bruce
Intelligent Robotics Group
National Aeronautics and Space Administration Ames Research Center

Spring semester 2017

Submitted on Aug. 18th, 2017

FACULTE SCIENCES ET TECHNIQUES DE L'INGENIEUR
BIOROBOTICS LABORATORY (BIOROB)
Station 9
CH-1015 LAUSANNE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MASTER PROJECT — Spring/Summer 2017			
Title:	**Manufacturing, Control and Testing of a Tensegrity Robot for Planetary Landing and Exploration**		
Candidate:	Marc André Leroy	**Section:**	Microengineering
Professor:	Auke Ijspeert	**Supervisor:**	Vytas SunSpiral
Assistants:	Florin Dzeladini, Massimo Vespignani and Jonathan Bruce		

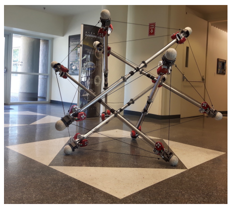
Abstract

The goal of this Master Project was to implement biologically-inspired control techniques for a tensegrity robot that will be used for planetary landing and exploration. Indeed, tensegrity structures are becoming increasingly popular as a dynamic architecture for robotics. As they are only composed of compressive elements and tensile elements, they have the potential of being lightweight, low-cost and more importantly, robust against impacts.

For these reasons these designs are being currently explored in the aerospace industry. Outer space missions require robots that would be capable of handling very harsh environments, and being able to reduce costs in what is a very costly industry is certainly of interest.

Thus, one could imagine using tensegrities as systems capable of Entry, Descent and Landing (EDL), as well as mobile systems once they are on the ground. What's more, the structure could be used as a protective cage for a scientific payload which would be safely secured within the structure.

The author of this report has had the extraordinary opportunity to work on such tensegrity systems at NASA Ames Research Center, specifically on the new version of the Spherical Underactuated Planetary Exploration Rover ball (SUPERball). This tensegrity robot is a tensegrity icosahedron composed of 6 rigid rods and 24 actuated cables.


SUPERball v2.0 prototype

However, previous versions of the robot only had 12 of its cables being actuated, the remaining 12 being passive. By having more actuators, it was thought that newer and more efficient locomotion control algorithms might be possible.

Central Pattern Generators (CPGs) and bio-inspired controls in general have proven to be simple models capable of generating very complex gaits in other types of robots, all thanks to their inherent robustness and modulation capabilities.

Trying to obtain locomotion out of the tensegrity icosahedra was the main research focus of this project. However, as there is currently no general methodology as to how one should implement these control algorithms into a physical system, different approaches were taken.

Firstly, a double chain of phase oscillators was implemented and tested thanks to the physical simulator known as NASA Tensegrity Robotics Toolkit (NTRT). Though some issues with the software collection were discovered, corrections were brought to the simulator. Nevertheless, some improvements are still required and are currently underway.

Secondly, a different approach was taken. By using Arbitrary Waveform Oscillators (AWOs) and amplitude-controlled CPGs, a framework to use CPGs as signal-learners was implemented and tested with locomotion algorithms obtained thanks to Reinforcement Learning with the 12-actuator version of SUPERball.

Finally, an implementation in Matlab was explored. It incorporates feedback elements into the CPGs thanks to the Tegotae rule, a Japanese concept which describes the extent to which a perceived reaction matches an expectation.

Parallel activities were also done throughout the project, such as contributing to the manufacturing of mockup models of the robot, testing of some of the mechanical components, and assembly of the final prototype of SUPERball v2.0.

Due to time constraints and hardware limitations with the current prototype, only some of the obtained locomotion algorithms could be tested on the physical prototype of the robot. Future works will include a comprehensive study of these algorithms in difficult terrain, as it is believed that the CPGs will be performant. Dynamic tensegrity structures are a relatively new field in research, but there is no doubt that new applications will be created in the very near future for these puzzling, yet compelling structures.

Acknowledgements

I would like to acknowledge first and foremost Prof. Auke Ijspeert, whom was the catalyst of my interest in bio-inspired robotics. The projects I did for his classes and for his lab's research allowed me to see that biology can bring much insight into how the world around us works. Not only that, robotics should take advantage of the results of millions of years of evolution, which should allow us to have better designed and more efficient robots in the near future.

My NASA supervisor, Vytas SunSpiral, should obviously also be thanked. He was the one who allowed me to go to NASA Ames Research Center in Mountain View, California to carry out the research required for the writing of this Master's thesis. It was such an extraordinary experience, where I was able to meet, work and laugh with so many great people, that I'll treasure the memories I've made there for all my life.

I would also like to acknowledge my NASA supervisors, Dr. Massimo Vespignani and Dr. Jonathan Bruce, whom taught me all that I now know about tensegrity structures and the advantages they bring into robot design. Thank you both as well for all the great tips on all the things there is to do in Califonia, which I only got to do a few, due to the limited time that I had there! What's more, they have also offered me to attend the 11th NASA/ESA Conference on Adaptive Hardware and Systems at the California Institute of Technology in Pasadena, California, which was also an incredible and extremely enriching opportunity.

Last but certainly not least, I would like to thank my EPFL assistant, Florin Dzeladini, who taught me much of what I know about Central Pattern Generators and their implementation. Even if he wasn't present at NASA, he would always have suggestions on how I could move forward when programming bugs blocked me.

Finally, I would like to mention some of the people I have collaborated with throughout this research, which include, but are not exclusive to Andrew Sabelhaus, Zakary Littlefield, Jeffrey Friesen, Brian Mirletz, Antoine Tardy, Natalie Ownby and Regina Apodaca. They have all inspired me, fueled many insightful discussions and support over the months I've spent in California.

This work is dedicated to my parents, Claudia and Pierre, and my sister Nathalie. Without their constant love, support and dedication, never would I have been able to make it to the end of this work. No words can express how much I am grateful to everything you ever taught and did for me! You all made me who I am today, and I couldn't wish for a better family!

List of Symbols and Abbreviations

List of Symbols

$\mathbf{A}$	Coupling matrix
A_0	Frequency offset in Fourier series
a_i and γ	Convergence factors
a_{ij} and w_{ij}	Coupling factor between oscillators i and j
a_n and b_n	1st and 2nd Fourier series factor
α	Amplitude factor
C	Cost function
c, K, k and σ	Coupling factor
d	Drive signal
e_{ij}	Intrinsic frequency difference between oscillators i and j
F and f	Arbitrary function
h	Coupling function
L	Set of neighbors
l	Current rest length
l_0	Initial rest length
μ	Amplitude-controlling parameter for an oscillator
N^V	Ground Reaction Force
N	Sensory information obtained through the environment or number of oscillators
n_L	Number of elements in L
ω_i	Intrinsic frequency of oscillator i
$\overrightarrow{\Omega}$	Intrinsic frequency matrix
P	Phase bias
ϕ_i	Phase of oscillator i
ϕ_{k_0}	Phase bias
ϕ_{ij}	Phase bias between oscillators i and j
ψ_{ij}	Phase difference between oscillators i and j
R_i	Oscillator intrinsic amplitude
r	Oscillator radius or amplitude state
$\overrightarrow{S}$	Matrix of sines of phase differences
s	Arbitrary signal
T	Tegotae function
t	Time
x	x-axis or arbitrary state
y	y-axis or arbitrary state
z	z-axis

List of Abbreviations

ARC	Ames Research Center
BioRob	Biorobotics Laboratory
BLDC	Brushless Direct Current
BPE	Bullet Physics Engine
CPG	Central Pattern Generator
DTRL	Dynamic Tensegrity Robotics Lab
EDL	Entry, Descent and Landing
GPS	Guided Policy Search
GRF	Ground Reaction Force
IRG	Intelligent Robotics Group
MDGPS	Mirror Descent Guided Policy Search
MLR	Mesencephalic Locomotor Region
NASA	National Aeronautics and Space Administration
NED	North East Distal
NEP	North East Proximal
NIAC	NASA Innovative Advanced Concept
NN	Neural Network
NTRT	NASA Tensegrity Robotics Toolkit
NWD	North West Distal
NWP	North West Proximal
OSG	OpenSceneGraph
PLA	PolyLactic Acid
PSO	Particle Swarm Optimization
ROS	Robot Operating System
SED	South East Distal
SEP	South East Proximal
SUPERball	Spherical Underactuated Planetary Exploration Rover Ball
SWD	South West Distal
SWP	South West Proximal
YAML	YAML Ain't Markup Language

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivations

Since ancient times, mankind has always aspired to explore the Universe out of our earthly bounds. Why are we here? What is our purpose? What should we accomplish in our lifetime? These are all questions that we believe could be answered (at least in part) thanks to outer space exploration, which finally started a few decades ago thanks to the hard work of engineers and scientists alike.

However, though many robotic missions on other celestial bodies have been successful, they all share one common factor: the robots used for planetary exploration are generally wheeled rovers, which are relatively simple to control, but not necessarily extremely robust to rough terrain and perturbations. In order to have robust locomotion, they need very large wheels that require greater torque or reduction gears, thus increasing the overall weight of the robot (as well as the cost of the mission). Moreover, this wouldn't prevent the robot from becoming stuck in soil, as was the case with the Spirit rover in Mars in 2009.

Human bipeds and other animals on the other hand are capable of extremely complex gaits, allowing them to achieve incredible locomotion, whether they're vertebrates or invertebrates. Why shouldn't we take more inspiration from ourselves when designing robots? Surely, taking inspiration from Nature itself would correspond to learning from millions of years of evolution and adaptation to harsh environments, which would surely be more than useful when exploring celestial bodies extremely inhospitable and different from our small Earth.

Ultimately, the goal of this Master Project was to contribute to the development of a novel type of robot for planetary landing and exploration, whose locomotion principles would be completely unusual in the development of space robotics.

What's more, the possibility of contributing to such a project within NASA Ames Research Center (NASA ARC) has truly been a once in a lifetime experience. Being able to help in the development of tools that will be used by the same agency that has had multiple historical achievements, such as the first human landing on another celestial body, and taking the first images from outside our Solar System, is something that the author of this report would like to show his unending gratitude towards those that have made this possible.

1.2 Goals

The author of this report has had the opportunity of carrying out his research within the Dynamic Tensegrity Robotics Lab (DTRL) of NASA ARC. Tensegrity robots, which will be properly presented in subsection 2.1.2, mimic structures similar to our own muscles and bones, thus creating robust skeletal structures able to adapt to different environments.

Tensegrity robots would offer a new approach into how rovers are designed, thanks to their inherent flexibility and large strength-to-weight ratio [1]. This is the reason why the Spherical Underactuated Planetary Exploration Rover ball (SUPERball) project has started in 2012 thanks to the funding awarded from NASA's Office of Chief Technologist for the NASA Innovative Advanced Concept[1] (NIAC).

In early 2017, the second physical prototype was being developed. The most significant change from the first one is that the second version has all 24 of its cables actuated, whereas the first one only had 12; the remaining 12 being passive. SUPERball v2.0 relies on motors that reel the cables in or out.

Thus, the primary goal of this report's author was to implement robust locomotion control algorithms by taking into consideration the unusual design of SUPERball. Then, if time allowed, map those commands into the full prototype, and test them.

The author was also able to help other researchers in minor tasks, such as prototyping and testing phases. This report focuses then on the main task and references some aspects and results of those minor tasks that were obtained between the months of February and August of 2017.

1.3 Contributions of this Master Project

Tensegrity structures being closely related to biological ones, it was of interest to experiment if Central Pattern Generators (CPGs) would allow to generate locomotion for a tensegrity icosahedron.

In the scope of the SUPERball project of the DTRL at NASA ARC, three different control architectures were implemented and tested to carry out this research:

- Using a double chain of phase oscillators (section 5.1);
- Using an arbitrary signal learner (section 5.2);
- Using a Tegotae-based approach (section 5.3).

Each one of these contributions shall be detailed in their respective sections. Other minor contributions were also done for the manufacturing of scaled mockup models, as well as the mechanical assembly and testing of the robot (chapter 6).

[1]https://www.nasa.gov/directorates/spacetech/niac/2012_phase_I_fellows_agogino.html

Chapter 2

State of the art

2.1 Tensegrity structures

2.1.1 Tensegrity structures' definition

As it is stated in the title of this report, the robot being developed in the DTRL is a tensegrity robot. A short explanation of their principle is presented in this section, as not many people are familiar with what it implies.

The term tensegrity has first been coined by the American architect Buckminster Fuller in the 1960s, whose meaning is "tensional integrity". Indeed, this design principle is characterized by combining compressive and tensional elements such that the different forces and moments are distributed along the whole structure, and not localized for example at joints, as is generally the case for rigid structures.

A simple method of creating a tensegrity structure is to have members whose load is purely (up to a certain extent) compressive or tensional, such as rigid rods and elastic cables. Thus, the only way the structure would be capable of failing would be if the cables yielded or if the rods buckled beyond repair. What's more, the cables are generally preloaded, such that they are rigid in tension.

Thanks to the specific arrangement of the structure's members that is generally used, only axial forces act upon it. There are indeed no bending nor shearing forces.

In order to illustrate this concept, the simplest tensegrity structure may be shown, namely the 3-rod prism, which as the name implies, is composed of three rigid rods, connected by six tensile cables, as it is shown in Figure 2.1. It can also be seen that the prism can rest on a stable position if three nodes are touching the ground.

There are theoretically no limitations as to how many rods or cables one uses when creating a tensegrity structure. There are however different classes of tensegrity structures. The 3-rod prism shown in Figure 2.1 for example is a Class-1 structure, as it corresponds to a structure

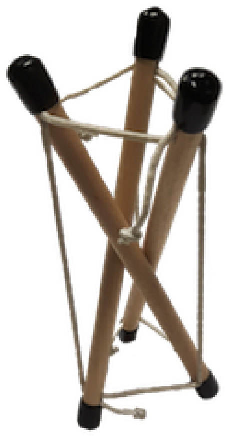


Fig. 2.1: 3-rod prism tensegrity structure

with only one rod connected to a given node without any additional constraint. Should there have been nodes where there are more than a single rod connected, or any additional constraints are characterized, then the class number increases. Figure 2.2 illustrates an example of a Class-2 tensegrity, as in some of the nodes there are two rods connected together.

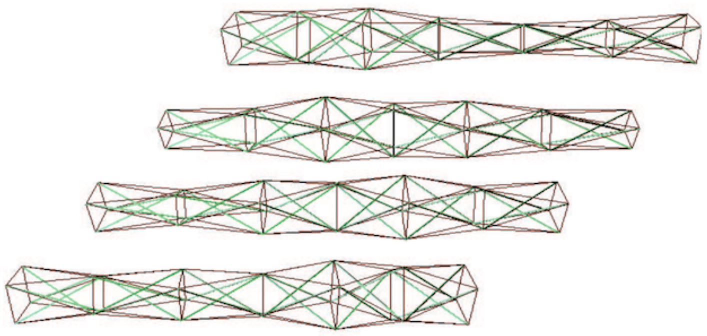


Fig. 2.2: Class-2 tensegrity example, a concept for an actuated tensegrity worm, taken from [2]

Tensegrity structures have steadily been gaining popularity as static constructions ever since the 1960s, but with recent advancements in control systems and actuator technologies, it is safe to assume that they are about to become more and more present in academia and industry as dynamic forms.

2.1.2 Advantages of tensegrity structures

Tensegrity structures being composed of rigid rods connected through a network of elastic cables, their main advantages compared to other rigid, "traditional" robot structures are their light weight, low cost, and inherent robustness to impacts thanks to their ability to deform and distribute forces across the entire structure.

What's more, tensegrities are largely present throughout Nature, which leads us to assume that they are the result of millions of years of evolution, and are thus a robust and efficient type of structure.

One example of such presence would be in living cells. It has been demonstrated that tensegrities can explain much of the complexity of pattern and structure that is observed within the cytoskeleton [3]. Alternatively, tensegrities can also model the interaction between our bones and fascia, which is the collection of connective tissues (muscles, tendons, ligaments, etc.) in our bodies. Figure 2.3 shows for example a tensegrity model of a human spine, showing how the vertebrae are able to "float" without touching, all thanks to the tensile network holding everything together.

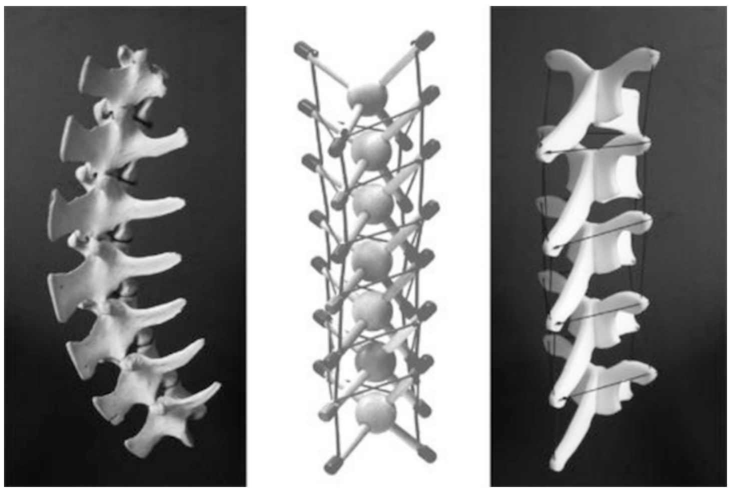


Fig. 2.3: (Left) Traditional skeleton model of the human spine, which incorrectly uses rigid hinge joints to connect vertebrae. (Center and right) Human spine models using the tensegrity principle where vertebrae are held together by soft elements (fascia), taken from [4]

This is also the reason why they are being studied as a new means to design robots, in particular space robots. Having a robot with a controllable tension network would allow it to be in a compact configuration, thus allowing tight stowage for launch, which would then be unpacked for landing once it arrives at its destination. In addition to it, a scientific payload would be protected, as the robot would act as an airbag by absorbing and distributing impact forces throughout the network of tensile elements. Along with the fact that locomotion could be achievable with dynamic tensegrity structures, the robot would continue being a protective cage for the payload throughout the scientific mission.

Having a robot that integrates Entry, Descent and Landing (EDL), as well as locomotion would largely decrease the mission's mass and cost budgets. This is what makes tensegrities particularly attractive to the aerospace industry.

2.2 SUPERball state of the art

As the author of this report arrived at NASA's lab a few years after the start of the SUPERball project, there was already a first prototype that had been built and preliminarily tested.

As the "U" in the acronym suggests, the first version of SUPERball was particularly underactuated, as only 12 of its 24 cables were actuated; the remaining 12 were passive. This would not only allow to simplify the model and the control aspect of the robot, but also serve as a proof of concept for moving tensegrity robots with the initial tests that would be carried out.

Figure 2.4 shows a picture of that first prototype. Actuation was done using a single motor

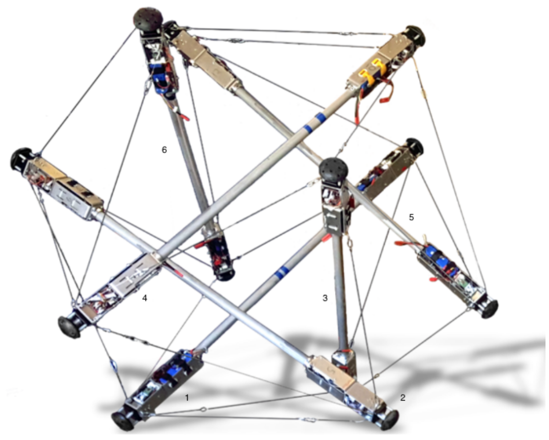


Fig. 2.4: SUPERball v1.0 prototype picture

per end cap, which was directly connected to a single string. By changing the cables' effective lengths, the tension in the corresponding cable would be changed. Springs contained within the rods provided additional compliance in the tension network.

Locomotion in v1.0 was achieved thanks to different control strategies, most notably after implementing deep reinforcement learning algorithms, which allowed the robot to achieve periodic gaits [5]. Results seemed very promising, as experiments demonstrated that it was possible to achieve a continuous locomotion gait for SUPERball v1.0's physical prototype.

However, as of spring of 2017, after many design reviews, SUPERball was being upgraded and it became possible to actuate all 24 of the structure's cables. This would surely allow new experiments in locomotion strategies, as well as efficiency. What's more, should there be any reason we would want to recreate scenarios from v1.0, turning off 12 of the actuators would make SUPERball v2.0 mostly similar to its predecessor. The physical prototype of v2.0 is shown in Figure 2.5.

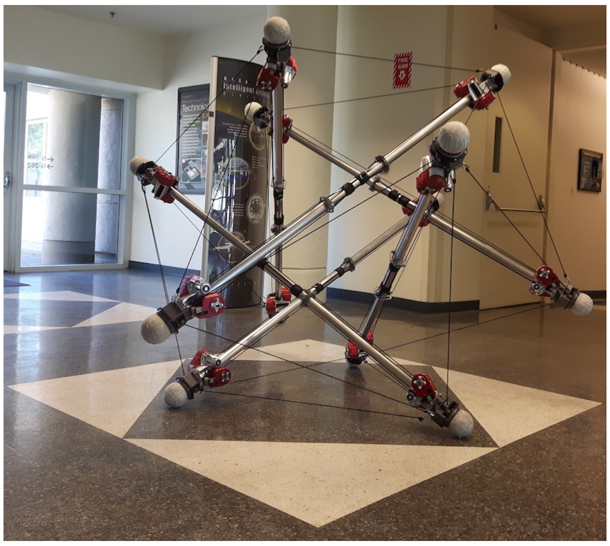


Fig. 2.5: SUPERball v2.0 physical prototype

The actuation in v2.0 will be done thanks to HEBI Robotics© X8-3 motors[1], which are torque-

[1]http://hebirobotics.com/products/

controlled BLDC motors. Each of the 24 motors is connected to a single cable in the structure. Subsequently, a torque input from the motor would generate a change in the cable's tension, which would correspond to a change in the effective length of the cable. Therefore, in simulation, one could model these control inputs either as motor torques or as changes in the cables' lengths.

It is assumed that Central Pattern Generators (CPGs) could be able to generate more efficient, robust and modulative locomotion capabilities for SUPERball v2.0, compared to the previously found strategies. These shall be detailed in chapter 4.

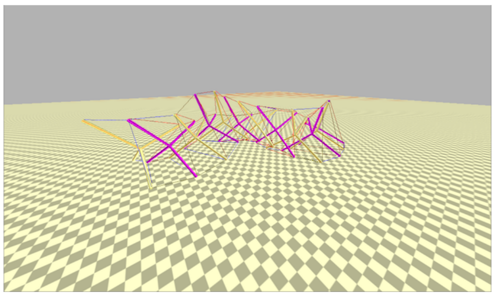
2.3 NASA Tensegrity Robotics Toolkit (NTRT)

2.3.1 Properties of NTRT

The NASA Tensegrity Robotics Toolkit is an evolving open-source collection of mostly C++ software modules used for the modeling, simulation and control of tensegrity robots. It was designed by the NASA ARC Intelligent Robotics Group (IRG) in order to enable rapid co-exploration of structures and control algorithms in a physics based environment built on top of the Bullet Physics Engine v2.82[2].

Much tensegrity research can be done thanks to this tool, and not only specifically to tensegrity icosahedra. The toolkit's architecture can indeed communicate directly with YAML files, which thanks to the fact that it is a human-readable data serialization format with a very simple and clean syntax, allows to create complex tensegrity structures in a considerably shorter and simpler way than what would be required in C++. The code in section A.1 shows for example how one would successfully implement the 3-rod prism in Figure 2.1.

Additionally, it does not require recompilation each time a modification to the structure is made, which allows for example to easily test different controllers with quite different structures. Other examples of tensegrity structures that can be built with YAML and NTRT can be found in Figure 2.6.


(a) Snake structure

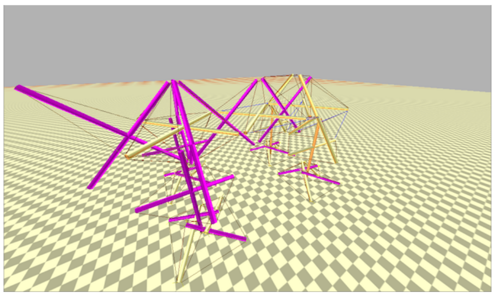

(b) Quadruped structure

Fig. 2.6: Different tensegrity structures deployed in NTRT

For the interested user, more information on how to install the toolkit may be found on http://ntrtsim.readthedocs.io/en/latest/setup.html. Additional information on how to get started with the toolkit is found on the same website, but should the new NTRT

[2]https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim

user want more details, an extensive doxygen documentation can be found in `http://ntrt.perryb.ca/doxygen/index.html`, whereas the NTRT's GitHub page is `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim`. All of these links have last been checked on August 17th 2017.

2.3.2 Architecture of an NTRT app

With NTRT correctly installed, it is possible to create the actual implementation. The main file is the app file, which is always created in a high level of abstraction and connects the tensegrity model to a simulated environment and a controller.

A world is first created (flat ground, slopes, hills, etc.), then graphics mode can yes or no be activated. Next, the tensegrity's model is attached to the environment, as well as the controller. The simulation is then finally started, and with a graphical output will continue until the user manually stops it, or can run for a specific amount of time if the graphics are deactivated.

An example of such a file is given in Appendix B. The implementation using YAML files instead of C++ model descriptions is the one showcased, as it was the methodology used throughout this project.

Chapter 3

Bio-inspired Control Theory

3.1 Dynamical systems

3.1.1 Limit cycles and their behavior

Many dynamical systems in the real world can present oscillations that can end up converging to a closed curve in a phase diagram. These isolated closed trajectories are what are called limit cycles. This means that all neighboring trajectories will spiral towards the limit cycle, in the case of a stable limit cycle.

Oscillators are therefore the category of systems that present this stable limit cycle behavior. One of the most well-known is the Van der Pol oscillator, which can have applications in electrical circuits, biology and other fields[1]. It is described by Equation 3.1.

$$\frac{d^2x(t)}{dt} - \mu \cdot (1 - x^2) \cdot \frac{dx(t)}{dt} + x(t) = 0 \tag{3.1}$$

Solving this equation gives indeed a limit cycle behavior, as it can be seen in Figure 3.1.

What's more, limit cycles are also capable of handling perturbations, as the system would return towards the closed trajectory independently of when the perturbation happens. The fact that some dynamical systems are capable of generating these periodic, robust oscillations indicates that it would be possible to achieve stable gait. This could thus be useful should the model be based on these oscillations.

Another example of well-known oscillator is the Hopf one, described by the set of equations in Equation 3.2, where μ controls its amplitude, ω its intrinsic frequency and r its radius.

[1] http://www.scholarpedia.org/article/Van_der_Pol_oscillator

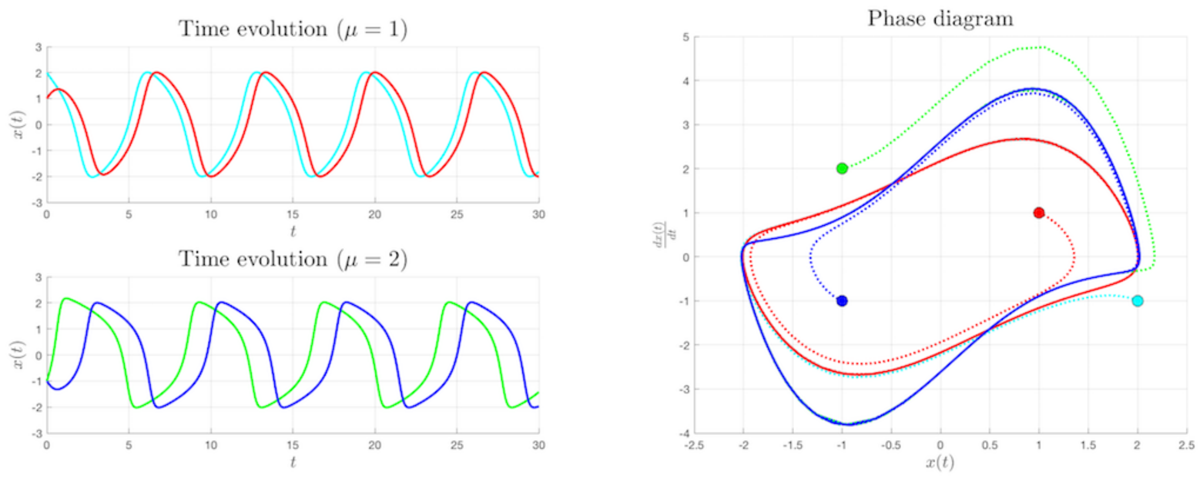


Fig. 3.1: Van der Pol oscillator illustration for different values of μ and initial conditions

$$\begin{cases} \frac{dx(t)}{dt} = (\mu - r^2) \cdot x - \omega \cdot y \\ \frac{dy(t)}{dt} = (\mu - r^2) \cdot y + \omega \cdot x \\ r = \sqrt{x^2 + y^2} \end{cases} \tag{3.2}$$

In the particular case of this oscillator, the geometrical angle in the phase diagram corresponds precisely to the oscillator's phase, which is not generally the case for other oscillators. The figure below illustrates the solution for this oscillator.

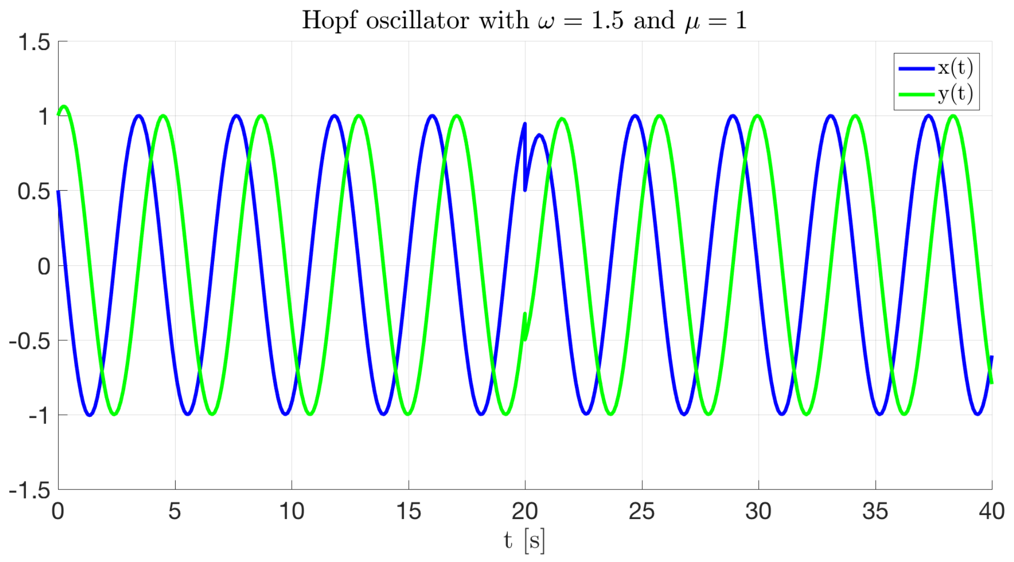


Fig. 3.2: Hopf oscillator solution, along with an added perturbation at time $t = 20[s]$

What's more, even though perturbations along the radius direction can be rejected, those in the phase direction are not. However, this is not necessarily harmful, as they could then be used to "reset" the oscillator's phase. This concept is thus the key to understanding synchronization effects, as well as many coordination phenomena in biology when multiple oscillators will be coupled in the next subsection.

3.1.2 Systems of coupled oscillators

The previous subsection briefly introduced the notion of dynamical systems that could present limit cycle behavior. It is widely accepted that many phenomena can be modeled thanks to coupled oscillators, which means that instead of relying on an equation of the type $\frac{dx(t)}{dt} = F_x(x)$, as was the case in Equation 3.1, two systems become interdependent, i.e. described by the set of equations of Equation 3.3, in the case of two coupled systems.

$$\begin{cases} \frac{dx(t)}{dt} = F_x(x, y) \\ \frac{dy(t)}{dt} = F_y(x, y) \end{cases} \tag{3.3}$$

Coupling nonlinear dynamical systems together introduces completely new and different dynamics, depending on the “strength” of the coupling. Still, with carefully selected couplings, the overall system’s output may be predicted by its subsystems properties. By using “weak” couplings, the subsystems should not change much, but obtain the added benefit of maybe synchronizing, i.e. getting a constant phase difference between the oscillators.

By coupling only two Hopf oscillators, very interesting behavior may already be achieved. The underlying general equations would be given by Equation 3.4.

$$\begin{cases} \frac{dx_1(t)}{dt} = (\mu_1 - r_1^2) \cdot x_1 - \omega_1 \cdot y_1 + K_1 \cdot x_2 + K_2 \cdot y_2 \\ \frac{dy_1(t)}{dt} = (\mu_1 - r_1^2) \cdot y_1 + \omega_1 \cdot x_1 + K_3 \cdot x_2 + K_4 \cdot y_2 \\ \frac{dx_2(t)}{dt} = (\mu_2 - r_2^2) \cdot x_2 - \omega_2 \cdot y_2 + K_5 \cdot x_1 + K_6 \cdot y_1 \\ \frac{dy_2(t)}{dt} = (\mu_2 - r_2^2) \cdot y_2 + \omega_2 \cdot x_2 + K_7 \cdot x_1 + K_8 \cdot y_1 \end{cases} \tag{3.4}$$

Rewriting these equations in polar coordinates would yield differential equations depending on radius and phase. As it has been stated that synchronization can be obtained if the phase difference is constant, standard fixed point analysis will be useful.

However, in order to simplify the understanding, let the chain of phase oscillators model presented in [6] be introduced in Equation 3.5, where ϕ_i is the phase of oscillator i, ω_i its intrinsic frequency and h a coupling function.

$$\frac{d\phi_i(t)}{dt} = \omega_i + h(\phi_{i-1}, \phi_i, \phi_{i+1}) \tag{3.5}$$

It actually corresponds to Hopf oscillators where the effects of radius are not taken into consideration. These have been previously used to model a chain of oscillators with exclusively nearest neighbor coupling and allowed to find analytical conditions indicating what was necessary in order to achieve synchronization. Indeed, Equation 3.6 presents the equations for this simplified nearest neighbor-coupled single chain of phase oscillators, where the coupling function is a sum of the sines of the phase differences, multiplied by a single common factor.

$$\begin{cases} \frac{d\phi_1(t)}{dt} = \omega_1 + a \cdot sin(\phi_2 - \phi_1) \\ \frac{d\phi_i(t)}{dt} = \omega_i + a \cdot sin(\phi_{i+1} - \phi_i) + a \cdot sin(\phi_{i-1} - \phi_i) \\ \frac{d\phi_N(t)}{dt} = \omega_N + a \cdot sin(\phi_{N-1} - \phi_N) \end{cases} \tag{3.6}$$

With these equations, the actual phase difference defined by $\psi_i = \phi_i - \phi_{i+1}$ can be introduced in order to rewrite Equation 3.6 as Equation 3.7.

$$\begin{gathered} \frac{d}{dt}\begin{bmatrix} \psi_1 \\ \dots \\ \psi_{N-1} \end{bmatrix} = \begin{bmatrix} \omega_1 - \omega_2 \\ \dots \\ \omega_{N-1} - \omega_N \end{bmatrix} + a \cdot \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \cdot \begin{bmatrix} \sin\psi_1 \\ \dots \\ \sin\psi_{N-1} \end{bmatrix} \\ \Updownarrow \\ \frac{d\vec{\psi}}{dt} = \vec{\Omega} + \mathbf{A} \cdot \vec{S} \end{gathered} \tag{3.7}$$

Finding a phase-locking condition corresponds therefore to finding conditions such that Equation 3.7 would be equal to 0, i.e. finding the $\vec{\psi}$ such that Equation 3.8 is satisfied.

$$\vec{S} = -\mathbf{A}^{-1} \cdot \vec{\Omega} \tag{3.8}$$

Still, it seems obvious that not all components of $\vec{S}$ will have the same magnitude. In order for the whole chain to synchronize, the equation above simply have to be mathematically feasible, i.e. matrix **A** should be invertible and all values of $\vec{S}$ should be within the unit circle. Thus, the phase-locking condition is dictated by the component of $\vec{S}$ with the highest magnitude.

In the case of a chain of N oscillators where we would have the same couplings as in Equation 3.6 and a constant frequency difference (i.e. $\vec{\Omega} = [e \dots e]^T$, $\vec{\Omega} \in \mathbb{R}^{N \times 1}$), the system would synchronize if Equation 3.9 is true.

$$\begin{cases} |\frac{e}{a}| \leq \frac{8}{N^2}, \text{ if } N \text{ is even} \\ |\frac{e}{a}| \leq \frac{8}{N^2-1}, \text{ if } N \text{ is odd} \end{cases} \tag{3.9}$$

Computer simulations illustrate the phase-locking behavior, as it can be seen in Figure 3.3.

It is evident that after a stabilization period the phase difference become constant, and the ensuing fixed points are highlighted by the circles on the right of Figure 3.3.

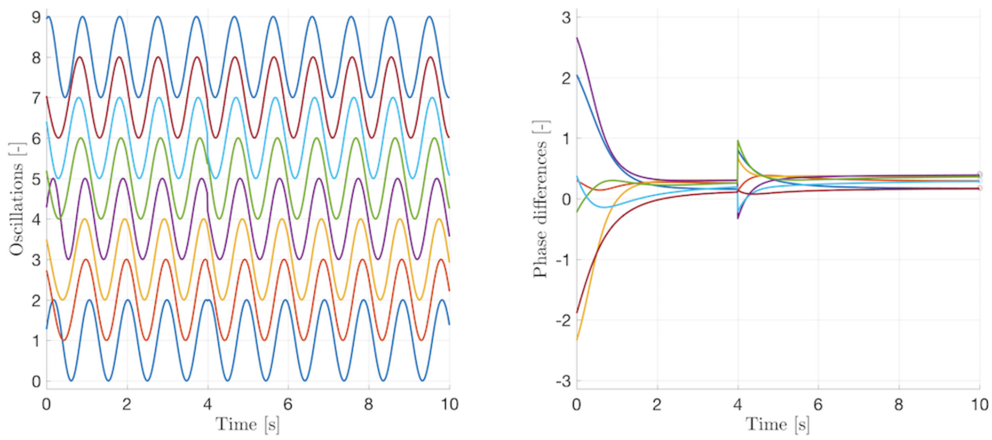


Fig. 3.3: Synchronization of single chain of phase oscillators. Circles to the right of the right figure indicate the fixed points for each oscillator. A perturbation was also added to illustrate the robustness of the limit cycle

3.1.3 Double Chain of Oscillators

The previous subsection showed that a single chain of oscillators is capable of achieving synchronization. However, some models may require higher complexity.

Biological evidence from [7] suggests that spines in vertebrates should in fact be modeled with a double chain of oscillators, thus increasing the complexity of the interactions of the oscillators one to another. Instead of being connected to one or two oscillators, as was the case in the single chain of the previous subsection, each oscillator in the double chain would be connected to either three or five other oscillators, as is illustrated in Figure 3.4.

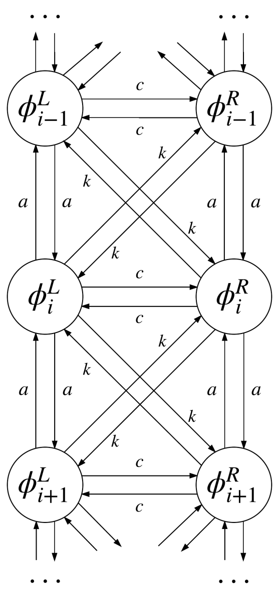


Fig. 3.4: Portion of double chain oscillators with equal couplings for different types of connections

This model is thus described by $2 \cdot N$ oscillators of phases ϕ_i^L (left) and ϕ_i^R (right), with i going from 1 to N. The illustration above also indicates that some of the model's parameters are set to equal values (coupling factors a, c and k), but this is only to simplify the model and underlying governing equations, which are therefore given by Equation 3.10.

$$\begin{cases} \dfrac{d\phi_i^L}{dt} = \omega_i + a \cdot \sin(\phi_{i-1}^L - \phi_i^L) + a \cdot \sin(\phi_{i+1}^L - \phi_i^L) \\ \qquad + k \cdot \sin(\phi_{i-1}^R - \phi_i^L) + k \cdot \sin(\phi_{i+1}^R - \phi_i^L) + c \cdot \sin(\phi_i^R - \phi_i^L) \\ \dfrac{d\phi_i^R}{dt} = \omega_i + a \cdot \sin(\phi_{i-1}^R - \phi_i^R) + a \cdot \sin(\phi_{i+1}^R - \phi_i^R) \\ \qquad + k \cdot \sin(\phi_{i-1}^L - \phi_i^R) + k \cdot \sin(\phi_{i+1}^L - \phi_i^R) + c \cdot \sin(\phi_i^L - \phi_i^R) \end{cases} \tag{3.10}$$

What's more, Rand, Cohen, and Holmes also added to the model a constraint forcing opposing sides of the spine to either be in-phase or out-of-phase, namely Equation 3.11.

$$\phi_i^R(t) = \phi_i^L(t) + P,\ P = 0 \text{ or } \pi \tag{3.11}$$

Equation 3.11 also has the added benefit of making that both left and right phase derivatives are equal, as indicated by Equation 3.12.

$$\frac{d\phi_i^L}{dt} = \frac{d\phi_i^R}{dt} \tag{3.12}$$

Equation 3.13 is obtained by substituting these in Equation 3.10.

$$\frac{d\phi_i^L}{dt} = \omega_i + (a \pm k) \cdot [\sin(\theta_{i-1}^L - \theta_i^L) + \sin(\theta_{i+1}^L - \theta_i^L)] \tag{3.13}$$

The last equation is actually of the same form as Equation 3.6. The $\pm$ sign comes from the choice of the value of P in Equation 3.11. Subsequently, the same concepts to find phase-locking conditions may also be used.

3.2 Central Pattern Generators

3.2.1 CPG basics

Locomotion control is a very difficult yet fundamental problem found everywhere in Nature, independently of which animal is trying to move. It indeed requires an efficient coordination of multiple degrees of freedom despite the many redundancies that may occur in the system, such as different end-point trajectories, postures, etc.

It has been shown that rythmic locomotion is achieved thanks to a central mechanism [8]. These experiments proved the existence of Central Pattern Generators (CPGs), which act as a feedforward component to the locomotion problem. Indeed, CPGs are neural networks that can produce rhythmic patterned outputs without rhythmic sensory or central input, observed both in vertebrate and invertebrate animals [9], [10].

Though CPGs could correspond to an open-loop control system, it has been shown that they are strongly modulated by sensory feedback [11]. Coupling both of together would thus allow us to obtain a robust closed-loop system, as illustrated in Figure 3.5.

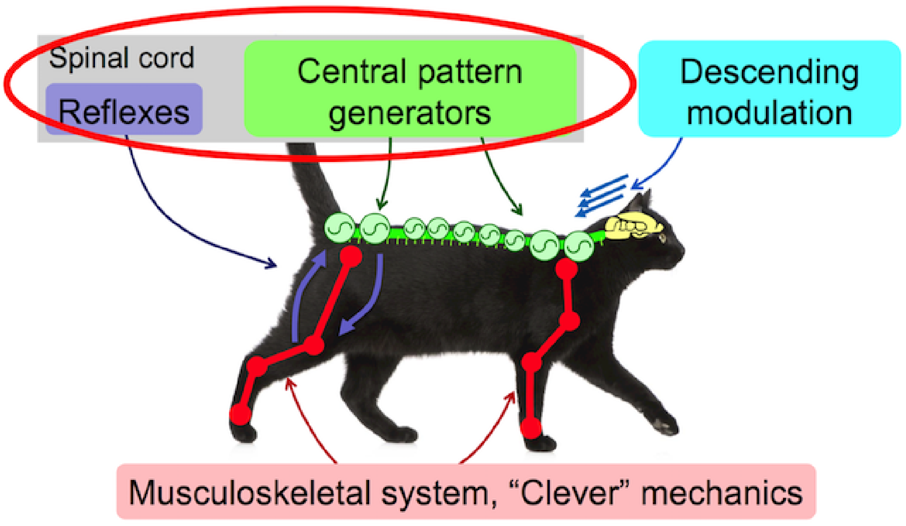

Spinal cord
Reflexes
Central pattern generators
Descending modulation
Musculoskeletal system, "Clever" mechanics

Fig. 3.5: Illustration of the different subsystems present in vertebrate locomotion[2]

In animal motor control this control strategy can therefore be described by four components, namely:

- A descending modulation signal coming from the brain, which corresponds to the input of the system;
- The CPGs, which transform the simple brain signals into the complex outputs that shall be transmitted to the rest of the animal;
- A musculoskeletal system, corresponding to the plant model;
- And finally the reflexes due to sensory inputs, thus being the feedback loop per say.

Seeing that animals are capable of extremely complex locomotion, it was only natural and a matter of time before robotics started to take inspiration from it. All items in the previous list may be transposed into components for a given robot. The modulation signal would be a driving signal generated by the robot's controller, which would then be transformed into different complex inputs thanks to the controller's logic, thus corresponding to the CPGs. The robot's mechanical structure naturally corresponds to the animal's musculoskeletal system and sensors embedded in the robot correspond to the sensory feedback.

Many robots have based their control strategy in CPG and sensory feedback control (e.g. [12], [13], [14]), and have showed promising results. It is therefore only natural that such a control strategy must be considered in the development of a mechanically and control-wise challenging robot such as SUPERball.

Therefore, these concepts shall be explored when implementing a control strategy for SUPERball in subsequent chapters of this report.

[2]Figure adapted from Computational Motor Control class, Lecture 5, Slide 32, École Polytechnique Fédérale de Lausanne, spring 2016

3.2.2 CPG Implementation

Arbitrary Wave-form Oscillators

As it was stated in the previous subsection, CPGs allow to convert simple drive signals into complex outputs. Still, there is no general methodology as to how one should implement CPGs. The only requirement is to obtain a limit cycle with a desired shape.

Ajallooeian *et al.* proposed a method of obtaining oscillators with an arbitrary limit cycle shape whose characteristics (frequency, amplitude, offset, shape and convergence) can be modulated online [15]. A simple implementation of their findings is through Arbitrary Wave-form Oscillators (AWOs).

Previous sections introduced the idea of a phase oscillator being a dynamical system described by Equation 3.5. One such example, where a phase bias ϕ_{ij} between oscillators i and j is introduced, would be to have:

$$\frac{d\phi_i(t)}{dt} = \omega_i + \sum_j \sin\left(\phi_j - \phi_i - \phi_{ij}\right) \tag{3.14}$$

The goal of AWOs is to use the phase values obtained with the equation above to drive them into a desired output $f_i(\phi_i)$. This is done thanks to the following equation describing an output x_i, where the first term corresponds to the feedforward driving term, the second one the feedback/attractor term (with γ_i determining the speed of convergence) and the final term K_i a random perturbation:

$$\frac{dx_i(t)}{dt} = \frac{df_i(\phi_i)}{d\phi_i} \cdot \frac{d\phi_i}{dt} + \gamma_i \cdot (f_i(\phi_i) - x_i) + K_i \tag{3.15}$$

An example of such a learned signal is given in Figure 3.6 and Figure 3.7. As it can be seen, even with perturbation, the phase bias was kept, thus illustrating the inherent robustness of the dynamic limit cycle.

In both figures, 12 signals were learned as a codyn framework was being implemented to learn the locomotion commands for SUPERball v1.0, as it shall be stated in section 5.2.

Seeing that in that test implementation each oscillator was coupled to one another, the phase biases ϕ_{ij} were not converging to the correct values; their mutual interaction modified them into other values. This was not the case if all biases were set to zero. That is the reason why both figures show all of them going to zero, as to overcome this issue, instead of making all $f_i = \cos(\phi_i)$, they were $f_i = \cos(\phi_i - 2 \cdot \pi \cdot {}^{(i-1)}/_{12})$.

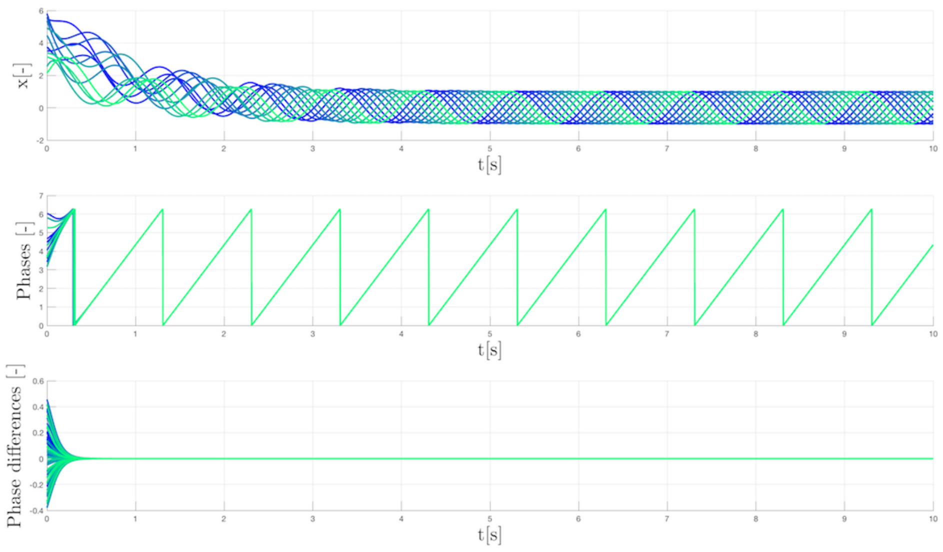


Fig. 3.6: AWOs learning 12 equally spaced cosine functions with frequencies equal to 1[Hz]

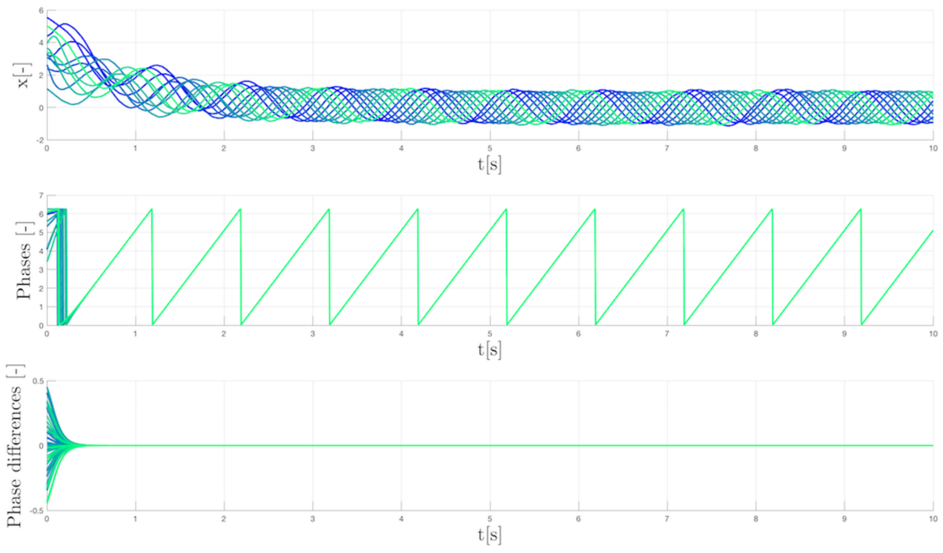


Fig. 3.7: AWOs learning 12 equally spaced cosine functions with frequencies equal to 1[Hz] and perturbations

Amplitude-Controlled CPGs

Another layer to the complexity of the CPG model. To ensure that smooth transitions are obtained during amplitude modulation, a critically damped second order linear differential equation may be included in the set of equations. This has indeed successfully reproduced animal locomotion thanks to a salamander robot [16]. The set of equations used in this model can thus be found in Equation 3.16.

$$\begin{cases} \frac{d\phi_i}{dt} &= \omega_i + \sum_j r_j \cdot w_{ij} \cdot \sin\left(\phi_j - \phi_i - \phi_{ij}\right) \\ \frac{d^2 r_i}{dt^2} &= a_i \cdot \left(\frac{a_i}{4} \cdot (R_i - r_i) - \frac{dr_i}{dt}\right) \\ x_i &= f_i(r_i, \phi_i) \end{cases} \tag{3.16}$$

The first equation is similar to Equation 3.14, though the amplitude state r_i of the oscillator and the coupling weight w_{ij} between oscillators i and j. The second one corresponds to the one controlling the smooth transitions, where R_i is the intrinsic amplitude to which the oscillator converges, r_i the amplitude variable and a_i a constant determining the speed of convergence for r_i to R_i. Finally, the third one corresponds to the arbitrary output which varies with r_i and ϕ_i , where in the case of [16] was set as $x_i = r_i \cdot (1 + cos(\phi_i))$ to represent the obtained burst signal. Solving these set of equations would result in what is shown in Figure 3.8.

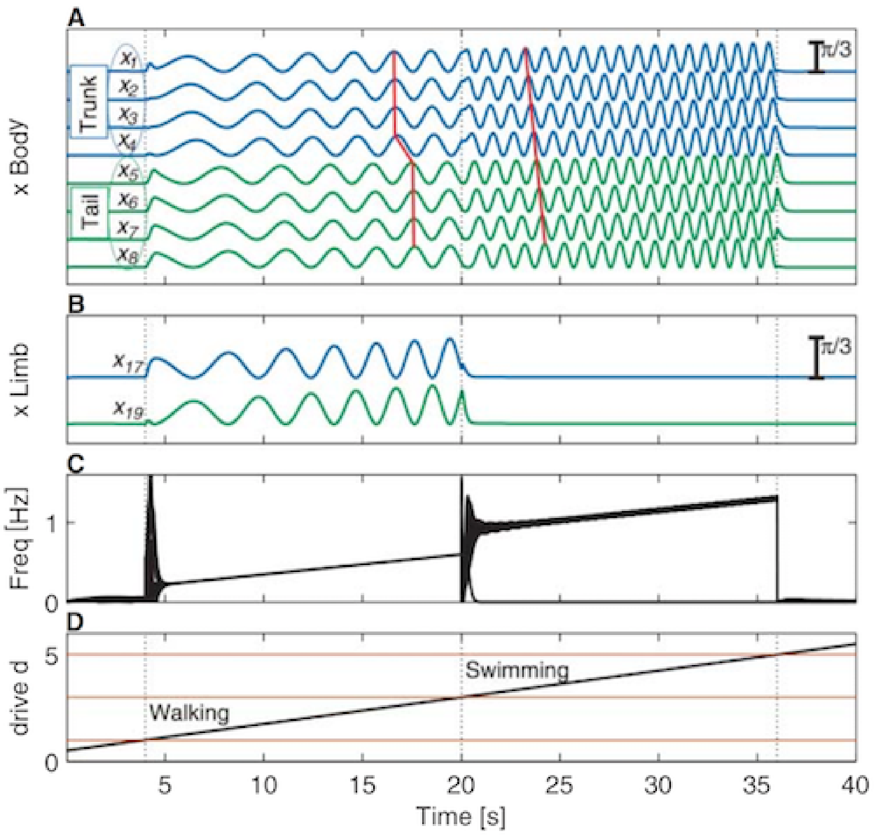


Fig. 3.8: Activity of the amplitude-controlled CPG model of the salamander robot in [16] when the drive signal is progressively increased. (A) x_i signals are body CPG oscillators. Units are in radians (scale bar on the top right). The red lines illustrate the transition from standing waves ($4[s] < t < 20[s]$) to traveling waves ($20[s] < t < 36[s]$). (B) x_i signals are limb CPG oscillators. Ipsilateral fore- and hindlimbs are in antiphase. (C) Instantaneous frequencies measured as [$^{cycles}/_s$]. The variations in the instantaneous frequencies among individual oscillators at times $t = 4[s]$ and $t = 20[s]$ correspond to brief accelerations and decelerations before resynchronization. (D) Linear increase of the drive d applied to all oscillators. The horizontal red lines correspond to the lower and upper oscillation thresholds for limb and body oscillators in arbitrary drive units, taken from [16]

This was done in order to be more closely related to the input signal coming from an animal's Mesencephalic Locomotor Region (MLR) of the brain [16]. This signal can indeed be modeled by a tonic drive that is capable of modulating the frequency and amplitude of specific oscillations between a lower and an upper threshold, such that some oscillators are react differently depending on the MLR's signal strength.

Indeed, saturation functions were also added in that robot, such that gait transitions were possible (from swimming to walking). One could therefore imagine using something similar to make SUPERball go smoothly from one type of gait to another, depending on which type of terrain it is currently located.

Tegotae-based approach

As it was illustrated in Figure 3.5, the CPGs are solely the feedforward component of the locomotion control algorithm. Though the CPG models previously discussed are inherently robust to perturbations thanks to their coupled nature, relying solely on feedforward models may not be enough depending on the desired application.

Owaki *et al.* have incorporated a feedback component within the minimal CPG model thanks to what they called the Tegotae approach. This Japanese concept describes the extent to which a perceived reaction matches an expectation [17]. In that article, Owaki *et al.* also explain this concept through a metaphor: if you want to lean against a wall, this corresponds to the intension of your physical controller, i.e. your nervous system; until the reaction force from the wall supporting your body is not enough, then “bad” Tegotae is obtained, whereas if it is sufficient, then “good” Tegotae is obtained. Subsequently, it is a connection between the perceived reaction received from the environment and the controller’s intention.

In order to quantify this Tegotae value, their model is related to the one of Equation 3.5, but instead of relying exclusively on neural coupling of the oscillators, a physical communication between limbs is modeled as in Equation 3.17.

$$\frac{d\phi_i(t)}{dt} = \omega_i + f_i = \omega_i + \frac{\partial T_i(\phi_i, N)}{\partial \phi_i} \tag{3.17}$$

In Equation 3.17, f_i corresponds to the local sensory feedback component incorporated to the oscillator model, T_i the Tegotae function per se, which is modulated by the phase/control variable ϕ_i and N represents the sensory information obtained through the environment.

The simplicity and effectiveness of this model has also been showcased in [17], where it was shown that interlimb coordination was achieved for a simple hexapod robot. Indeed, in their investigation, they found that even a simple Tegotae function where both ϕ_i and N are separated produced nice results. The Tegotae function used by them was the one described in Equation 3.18.

$$\begin{aligned} T_i(\phi_i, N) &= \sigma_1 \cdot T_{i,1}(\phi_i, N) + \sigma_2 \cdot T_{i,2}(\phi_i, N) \\ &= \sigma_1 \cdot \Big(-\sin(\phi_i) \cdot N_i^V \Big) + \sigma_2 \cdot \sin(\phi_i) \cdot \left(\frac{1}{n_L} \cdot \sum_{j \in L(i)}^{n_L} k_j \cdot N_j^V \right) \end{aligned} \tag{3.18}$$

Equation 3.18 consists of two linearly coupled Tegotae functions thanks to the coupling factors σ_1 and σ_2. What’s more, $T_{i,1}$ quantifies Tegotae on the corresponding leg thanks to vertical ground reaction forces (GRFs) N_j^V whereas $T_{i,2}$ that on the neighboring legs. This is coded into the equation thanks to the set $L(i)$, with n_L elements in it. Figure 3.9 illustrates this set in the case of a hexapod robot.

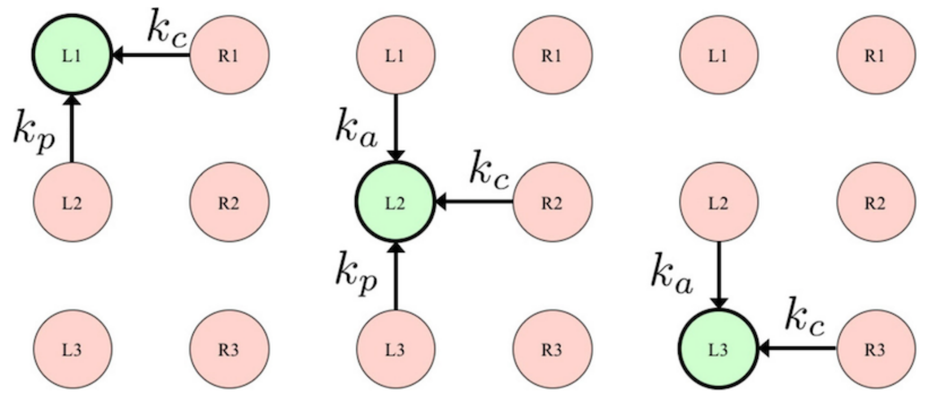


Fig. 3.9: Definition of $L(i)$, describing a set consisting of the legs neighboring the ith leg. The left, center, and right figures show the set of left fore ($L1$), left middle ($L2$), and left hind ($L3$) legs, respectively; k_j (k_a, k_p, $k_c \geq 0$) denotes the weight for each GRF N_j, taken from [17]

It will therefore be interesting to see if trying to do the same with SUPERball will also be able to generate some type of periodic gait. To the best of the author's knowledge, there is no animal in Nature which can directly be represented by a tensegrity icosahedra. Subsequently, there is no guarantee that this implementation shall work, but this should be tested either way.

Chapter 4

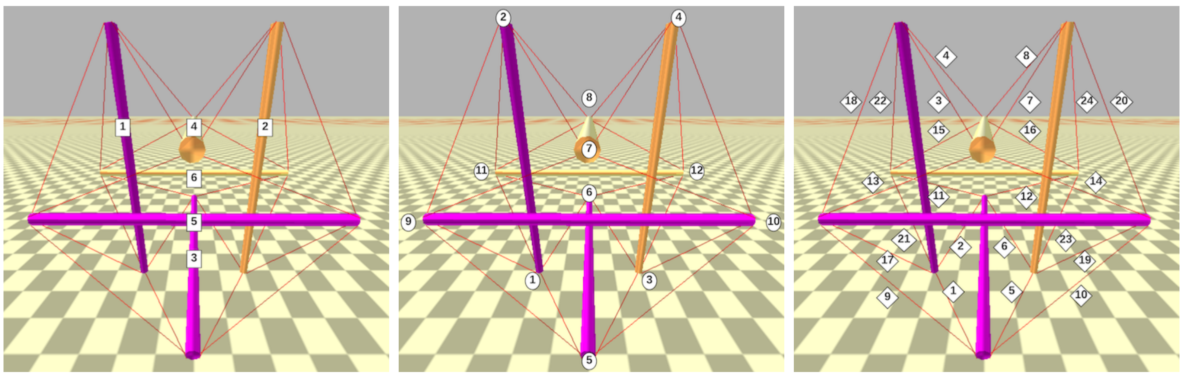
Controller Implementation

4.1 SUPERball Nomenclature

4.1.1 Nomenclature within NTRT

As computer simulations play a large role in this project, having a well established nomenclature that accurately distinguishes each rod and cable is of extreme importance.

Figure 4.1 contains a front view of SUPERball. Nevertheless, the YAML language would allow for the model to be dispatched in NTRT in any rotation configuration.



(a) Rods **(b)** End Caps **(c)** Actuators

Fig. 4.1: Established nomenclature for SUPERball v2.0

Each coded number is useful as they serve different purposes. Firstly, each rod is assigned to a number. Secondly, the ones of each end cap of each rod are distinguished so the YAML file correctly attaches a cable between two of them. Finally, each cable has its own code number as it is better to keep independent, higher-level control for each of them, and then merge some of them together, should we so desire.

What's more, this nomenclature was established with the physical prototype in mind, as it shall be explained in subsection 4.1.2.

4.1.2 Nomenclature within the physical prototype

During the time that the robot's controller's implementation was being done, the physical prototype was being built. In order to ease the transition from the simulation environment to the actual robot, using the same nomenclature was desired.

Each rod was numbered so that in a stable position they are in pairs according to the plane in which they are located. What's more, as each of the six rods of the robot supports four BLDC motors, every single one of them corresponds to one of the actuated cables within NTRT. For simplicity, each end cap of each rod would thus be hosting consecutive motor numbers.

Therefore, if one knows what rod is the one of interest, the end caps and motors can easily be deduced thanks to the representation given in Figure 4.2.

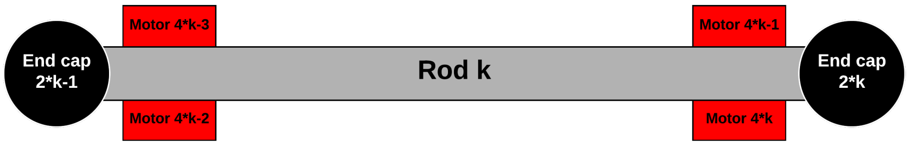


Fig. 4.2: Established nomenclature with physical rod of SUPERball v2.0

With this nomenclature, connection matrices can be defined, namely one representing the connection between each of the actuators and one representing which actuators are connected to a rod's end cap. A graphical representation of both of these can be seen in Figure 4.3.

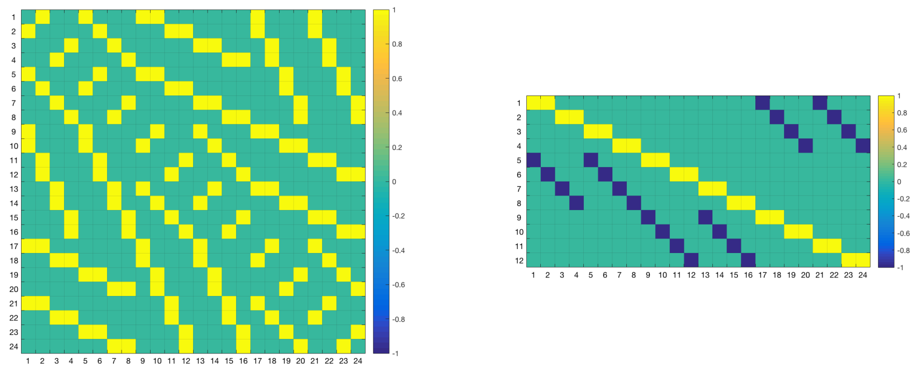


Fig. 4.3: Graphical representation of the connection matrices of SUPERball with the current nomenclature. Left corresponds to neighboring actuators and right corresponds to actuators connected to an end cap (1 if motor at that end cap and -1 if at another)

4.2 Preliminary Model Considerations

SUPERball being an icosahedron, the end caps at the rods' ends are the elements that shall generally be touching the ground. What's more, the fact that three nodes would be on the ground on a stable configuration makes that the formed base triangle is the one keeping the structure static.

Different methods could be explored to achieve locomotion. For example, one could try to make two of the nodes come closer together, such that the base shape would no longer be a triangle, but a line, thus being a non-stable configuration, which would make the robot tip over. Alternatively, if the base triangle keeps its shape but reduces its surface area, and other ones increase theirs, it can be imagined that the center of gravity would move over time, thus making the less stable, smaller base triangle get off the ground. It can easily be imagined that a triangle with a larger surface area would go into contact with the ground. Consequently, repeating this process should allow some type of locomotion generation.

Subsequently, SUPERball's locomotion controller should be implemented by taking into consideration these stable triangles. Not only would they serve as stable intermediate points during its path, but it would also allow to use the controller implemented for this report in future tensegrity robots, whether they are icosahedra or any other structure having triangular faces (e.g. icododecahedra).

Another consideration that should not be forgotten is that the motors are not perfect. There is indeed a limit as to how quickly they are able to turn. When the physical prototype was built, tests with the motors showed that at maximum speed they were able to spool out approximately $5[cm]$ of cable in $1[s]$. This was included in the control loops that were implemented for chapter 5.

4.3 Torque Control

As it was stated in section 2.2, the motors being integrated in v2.0 of SUPERball are the X8-3s from HEBI Robotics©. Seeing that these are torque-controlled motors, it would be beneficial if all control simulations could be done in accordance to it, such that the mapping of control inputs into the actual prototype will be more easily done.

It was thus decided to use the `tgKinematicActuator` class of NTRT, thanks to its implementation of important motor characteristics, such as inertia and friction. What's more, the implementation is such that the control inputs sent will be directly assigned to the desired torque value of a given actuator.

By implementing SUPERball's controller in such a manner would surely make hardware testing easier and in more accordance with what would be output by the computer simulations. However, after some preliminary basic controller implementations within NTRT, some issues arose.

Specifically, in the class' functions, a motor's acceleration was being computed thanks to its physical characteristics and state at a specific timestamp. It would then be numerically integrated in order to deduce velocity, thus changing the effective length of the elastic member it was attached to in the tensegrity structure.

The problem seemed to be originating from accumulator errors, as turning off a controller after a few seconds released an incredible amount of tension, thus making SUPERball "jump" from the ground, which was physically impossible.

Other users and software developers currently working on NTRT have been notified, and there is ongoing work to correct these issues. However, as of the writing of this report, no success has been found so far as to what exactly could be causing the problem. Still, as the simulator was built on top of the Bullet Physics Engine (BPE), it may be possible that it was an error inherent to BPE, considering that it is first and foremost a video game engine. Another method for the implementation of the controller should then be used.

4.4 Position Control

The issues with the `tgKinematicActuator` class presented in section 4.3 made it necessary to revert to a simpler, less accurate model, namely the one from the `tgBasicActuator` class. This class unfortunately does not provide with any sort of friction or inertia of a given motor, as they are more closely related to the elastic members.

Still, it has the added benefit of having been used extensively in previous models built on NTRT, and not only specifically for SUPERball. What's more, it should also be noted that previous research that had hardware successfully validated was based on control strategies based off of this class [18].

In this class, the control inputs are directly sent to the cables themselves, directly changing the effective length of the member. Surely, when it would be time to map it to v2.0's prototype, some hand tuning would inevitably be required due to the inaccuracies of the model; but should the controller be trained with some disturbances, it may take into account the model's errors, thus making the transition simulations-prototype less harsh.

4.5 Preliminary Tests

4.5.1 General considerations

SUPERball contains 24 motors actuating 24 elastic cables. The control strategy should thus be able to coordinate all of them simultaneously. However, it is known that one of the kinematically stable positions for SUPERball is when it is resting in one of the eight triangles formed by three elastic cables [19]. These are illustrated in Figure 4.4.

One of the methods to be explored would then be considering that each of the robot's triangles is a single oscillator. This means that all three cables of the same triangle would receive the same control inputs from each of their corresponding motors, be it for shortening the effective length or to give the cables more slack. It was thought that this method could give feasible results, as it could be programmed that if one of the triangle shrinks, the neighboring ones extend and vice versa.

Thus, neighboring triangles should be in anti-phase. The fact that some triangles may be increasing their area (thus their tension) means that other ones should be decreasing their corresponding ones, in order not to create too much of a force gradient in the prototype, which could be hazardous.

What's more, seeing that SUPERball v2.0 is being designed such that from the resting position

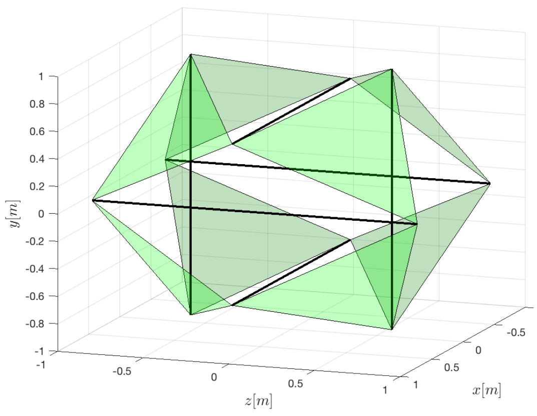
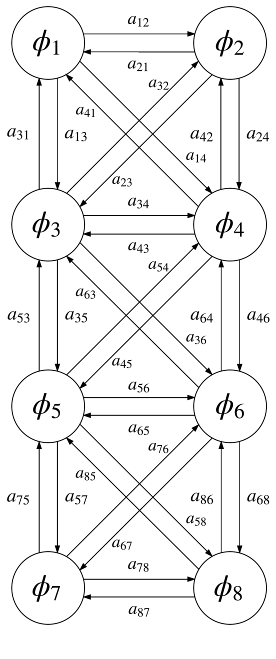


Fig. 4.4: Simplified representation of SUPERball. Combining three cables together gives one of the eight kinematically stable triangles on top of which the robot may rest. Darker shades of green represent the triangles on the background, whereas the lighter ones represent the ones on the foreground

each motor shall be capable of reeling in or releasing a rod-length of cable ($\approx 2[m]$), the oscillations generated by the CPGs will be able to have a large amplitude, which could end up allowing SUPERball to have greater stride lengths, thus being able to roll faster.

4.5.2 SUPERball as a double chain of phase oscillators

The biological models first presented in subsection 3.1.3 presented reliable means of modeling a spine, which was capable of achieving phase-locked behavior. SUPERball being a rolling robot, a regular, synchronized actuation pattern should be investigated. This is the reason why each of its eight triangles illustrated in Figure 4.4 is being assigned one of phase oscillators of Figure 3.4. The complete chain is thus illustrated in Figure 4.5.



ϕ_1
ϕ_2
ϕ_3
ϕ_4
ϕ_5
ϕ_6
ϕ_7
ϕ_8
a_{12} a_{21} a_{32} a_{41} a_{31} a_{13} a_{42} a_{24} a_{14} a_{23} a_{34} a_{43} a_{54} a_{63} a_{53} a_{35} a_{64} a_{46} a_{36} a_{45} a_{56} a_{65} a_{76} a_{85} a_{75} a_{57} a_{86} a_{68} a_{58} a_{67} a_{78} a_{87}

Fig. 4.5: SUPERball modeled as a double chain of phase oscillators. Each oscillator corresponds to a group of three actuated cables

This simplification in the model means however that the chain of oscillators has a “beginning” and an “end”. Indeed, many different models were attempted such that this would not be the case, but the ensuing equations would not be capable of generating synchronized behavior. The subsequent matrix **A** in Equation 3.8 would indeed not be invertible.

This thus creates a discontinuity in the model, but that was assumed to be non-critical at this point of the project. What’s more, it was believed that this model would be able to generate locomotion in a given direction. Subsequent implementations could thus explore this “triangle-mapping” and change it in order to change directions. For example, should a given mapping generate forward locomotion, inverting the chain’s oscillators’ order would allow SUPERball to move backwards.

However, as SUPERball does not correspond exactly to a spine, it was desired to have a model with more complexity than the one presented in subsection 3.1.3. Couplings would indeed not necessarily be symmetrical, thus yielding Equation 4.1, where $\psi_{ij} = \phi_i - \phi_j$.

$$\begin{cases} \dot{\phi_1} = \omega_1 - a_{21} \cdot \sin\psi_{12} - a_{31} \cdot \sin\psi_{13} - a_{41} \cdot \sin\psi_{14} \\ \dot{\phi_2} = \omega_2 + a_{12} \cdot \sin\psi_{12} - a_{32} \cdot \sin\psi_{23} - a_{42} \cdot \sin\psi_{24} \\ \dot{\phi_3} = \omega_3 + a_{13} \cdot \sin\psi_{13} + a_{23} \cdot \sin\psi_{23} - a_{43} \cdot \sin\psi_{34} - a_{53} \cdot \sin\psi_{35} - a_{63} \cdot \sin\psi_{36} \\ \dot{\phi_4} = \omega_4 + a_{14} \cdot \sin\psi_{14} + a_{24} \cdot \sin\psi_{24} + a_{34} \cdot \sin\psi_{34} - a_{54} \cdot \sin\psi_{45} - a_{64} \cdot \sin\psi_{46} \\ \dot{\phi_5} = \omega_5 + a_{35} \cdot \sin\psi_{35} + a_{45} \cdot \sin\psi_{45} - a_{65} \cdot \sin\psi_{56} - a_{75} \cdot \sin\psi_{57} - a_{85} \cdot \sin\psi_{58} \\ \dot{\phi_6} = \omega_6 + a_{36} \cdot \sin\psi_{36} + a_{46} \cdot \sin\psi_{46} + a_{56} \cdot \sin\psi_{56} - a_{76} \cdot \sin\psi_{67} - a_{86} \cdot \sin\psi_{68} \\ \dot{\phi_7} = \omega_7 + a_{57} \cdot \sin\psi_{57} + a_{67} \cdot \sin\psi_{67} - a_{87} \cdot \sin\psi_{78} \\ \dot{\phi_8} = \omega_8 + a_{58} \cdot \sin\psi_{58} + a_{68} \cdot \sin\psi_{68} + a_{78} \cdot \sin\psi_{78} \end{cases} \tag{4.1}$$

What’s more, Equation 4.1 may be rewritten as Equation 4.2 by remembering the constraint imposed by Equation 3.11, the latter of which also makes that $\dot{\phi}_{2\cdot n-1} = \dot{\phi}_{2\cdot n}$, $n = 1, 2, 3, 4$. There is also a benefit in using this constraint: all of the oscillators are in anti-phase with their respective neighbors. Subsequently, even if there are technically discontinuities in the chain model, the starting and ending oscillators are also in anti-phase. Subsequently, there shouldn’t be high tension discontinuities, which could be harmful for the robot.

$$\begin{cases} \dot{\phi_1} = \omega_1 - (a_{31} \pm a_{41}) \cdot \sin\psi_{13} \\ \dot{\phi_3} = \omega_2 + (a_{13} \pm a_{23}) \cdot \sin\psi_{13} - (a_{53} \pm a_{63}) \cdot \sin\psi_{35} \\ \dot{\phi_5} = \omega_3 + (a_{35} \pm a_{45}) \cdot \sin\psi_{35} - (a_{75} \pm a_{85}) \cdot \sin\psi_{57} \\ \dot{\phi_7} = \omega_4 + (a_{57} \pm a_{67}) \cdot \sin\psi_{57} \end{cases} \tag{4.2}$$

The differential equations characterizing the phase difference are finally stated in Equation 4.3, where $e_{ij} = \omega_i - \omega_j$.

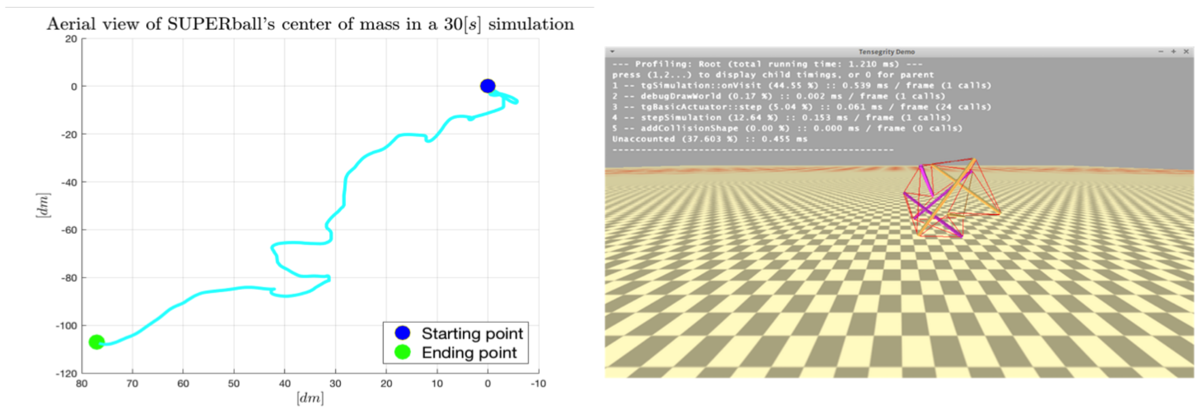
$$
\begin{gathered}
\begin{cases}
\dot{\psi}_{13} = e_{13} - (a_{31} \pm a_{41} + (a_{13} \pm a_{23})) \cdot \sin\psi_{13} + (a_{53} \pm a_{63}) \cdot \sin\psi_{35} \\
\dot{\psi}_{35} = e_{35} + (a_{13} \pm a_{23}) \cdot \sin\psi_{13} - (a_{53} \pm a_{63} - (a_{35} \pm a_{45})) \cdot \sin\psi_{35} + (a_{75} \pm a_{85}) \cdot \sin\psi_{57} \\
\dot{\psi}_{57} = e_{57} + (a_{35} \pm a_{45}) \cdot \sin\psi_{35} - (a_{75} \pm a_{85} + (a_{57} \pm a_{67})) \cdot \sin\psi_{57}
\end{cases} \\
\Updownarrow \\
\frac{d\vec{\psi}}{dt} = \begin{bmatrix} e_{13} \\ e_{35} \\ e_{57} \end{bmatrix} + \begin{bmatrix} -a_{31} \mp a_{41} - a_{13} \mp a_{23} & a_{53} \pm a_{63} & 0 \\ a_{13} \pm a_{23} & -a_{53} \mp a_{63} + a_{35} \pm a_{45} & a_{75} \pm a_{85} \\ 0 & a_{35} \pm a_{45} & -a_{75} \mp a_{85} - a_{57} \mp a_{67} \end{bmatrix} \begin{bmatrix} \sin\psi_{13} \\ \sin\psi_{35} \\ \sin\psi_{57} \end{bmatrix} \\
\Updownarrow \\
\frac{d\vec{\psi}}{dt} = \vec{\Omega} + \mathbf{A} \cdot \vec{S}
\end{gathered}
\tag{4.3}
$$

An equation exactly similar to that of Equation 3.7 is obtained. If this new matrix $\mathbf{A}$ is invertible, then the overall system will synchronize, i.e. SUPERball will achieve periodic, constant phase-difference oscillations for each of its triangles. With this, preliminary results can be obtained. To do so, the cables were commanded as in Equation 4.4, where l_i is the new cable rest length command, l_{i_0} its initial rest length and ϕ_{i_0} the phase offset P of Equation 3.11 of oscillator i.

$$
l_i(t) = l_{i_0} \cdot \left(1 + \frac{1}{2} \cdot \cos\left(\phi_i(t) + \phi_{i_0}\right)\right) \tag{4.4}
$$

The ϕ_{i_0} were set to 0 for $i = 1, 4, 5, 8$ and π for $i = 2, 3, 6, 7$. The reason why this was done was that unlike the lamprey application used in [6], it wouldn't make sense for SUPERball to have one side of the chain having phases too close one to another. Surely, as each oscillator corresponds to a triangle where three cables retract or extend simultaneously, the three neighboring triangles should generally be having the opposite behavior. This would indeed prevent high tensions at one of the robot's nodes, which could ultimately snap one of the cables.

One of these results can be seen in Figure 4.6, where random values that fulfilled the conditions above were assigned to the controller.



```
--- Profiling: Root (total running time: 1.210 ms) ---
press (1,2...) to display child timings, or 0 for parent
1 -- tgSimulation::onVisit (44.55 %) :: 0.539 ms / frame (1 calls)
2 -- debugDrawWorld (0.17 %) :: 0.002 ms / frame (1 calls)
3 -- tgBasicActuator::step (5.04 %) :: 0.061 ms / frame (24 calls)
4 -- stepSimulation (12.64 %) :: 0.153 ms / frame (1 calls)
5 -- addCollisionShape (0.00 %) :: 0.000 ms / frame (0 calls)
Unaccounted (37.603 %) :: 0.455 ms
-------------------------------------------------
```

Fig. 4.6: Results of the double chain of oscillators model with random values assigned to the controller's parameters, where $\frac{d\vec{\psi}}{dt} = \vec{0}$

Though it seems that SUPERball followed a general direction, the overall movement was not fully straight nor periodic. Optimization of the open parameters should allow better results to be obtained. This will be detailed in section 5.1.

4.5.3 Mapping the oscillators to different triangles

As it was previously mentioned, it is assumed that by using the double chain of phase oscillators as defined in Figure 4.5 would allow SUPERball to move in a specific direction. Using the front view of Figure 4.1 as a reference and by using the acronyms N for North, S for South, E for East, W for West, P for Proximal and D for Distal, one way one could map the actuators triangles is given in Figure 4.7.

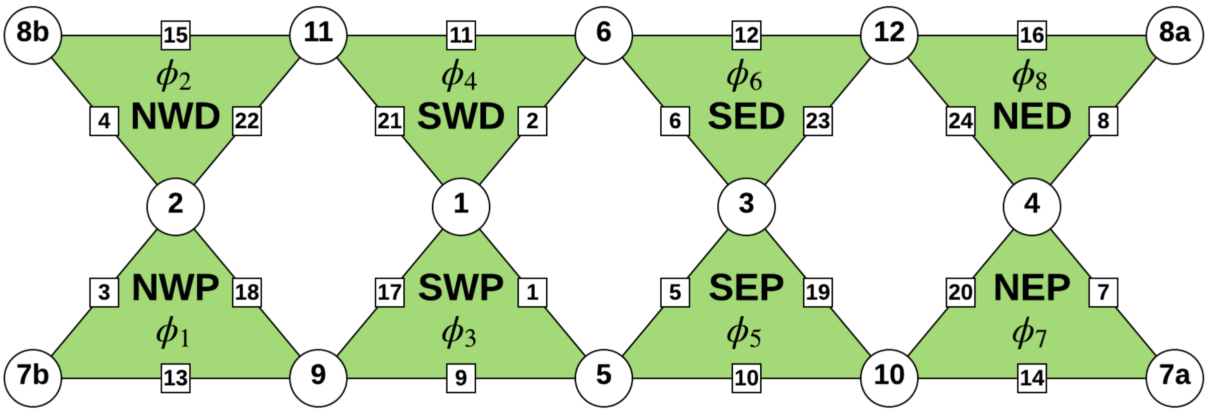


Fig. 4.7: 1st example of how one could map SUPERball's triangles as a double chain of oscillators

If for example that would generate forward, it can be assumed that the 2nd mapping given in Figure 4.8 would be able to generate locomotion towards the left, all thanks to the symmetrical structure of the icosahedra.

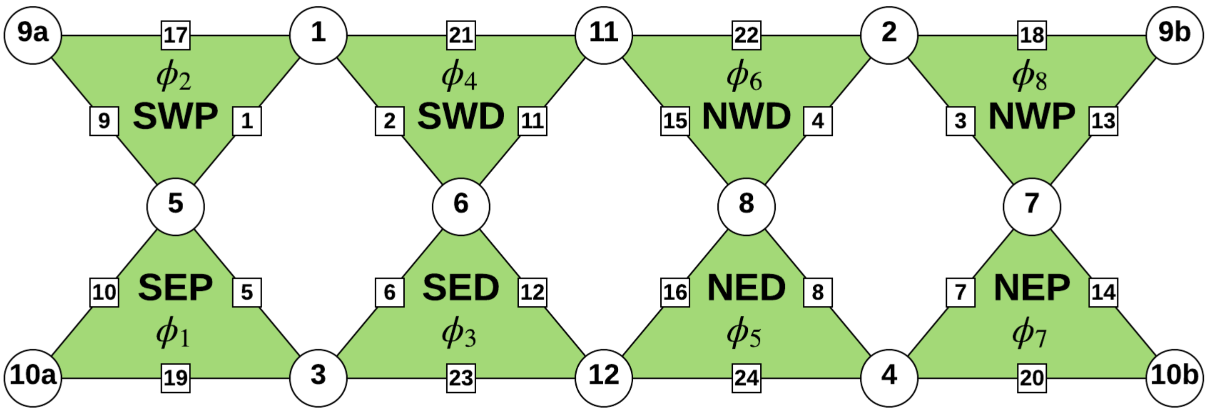


Fig. 4.8: 2nd example of how one could map SUPERball's triangles as a double chain of oscillators

Tests in section 5.1 shall indicate if these hypotheses are held or not.

4.6 Tuning the Controller

The large number of open parameters within the chain of phase oscillators model makes it clear that hand tuning them would not be the most effective method of obtaining a successful controller. Therefore, it would be wiser to use learning algorithms to drive the parameters towards these practical values.

Machine learning algorithms had started to be incorporated within NTRT in the form of a learning library of functions that may be called within the general call of an app file[1]. A few different options were available, such as Monte Carlo, Gaussian, Co-evolutionary or Genetic algorithms.

Incorporating this into the NTRT app file can also easily be done. As the library contains functions that have to be called within the app file, this would change the process flow of the simulation into what is shown in Figure 4.9.

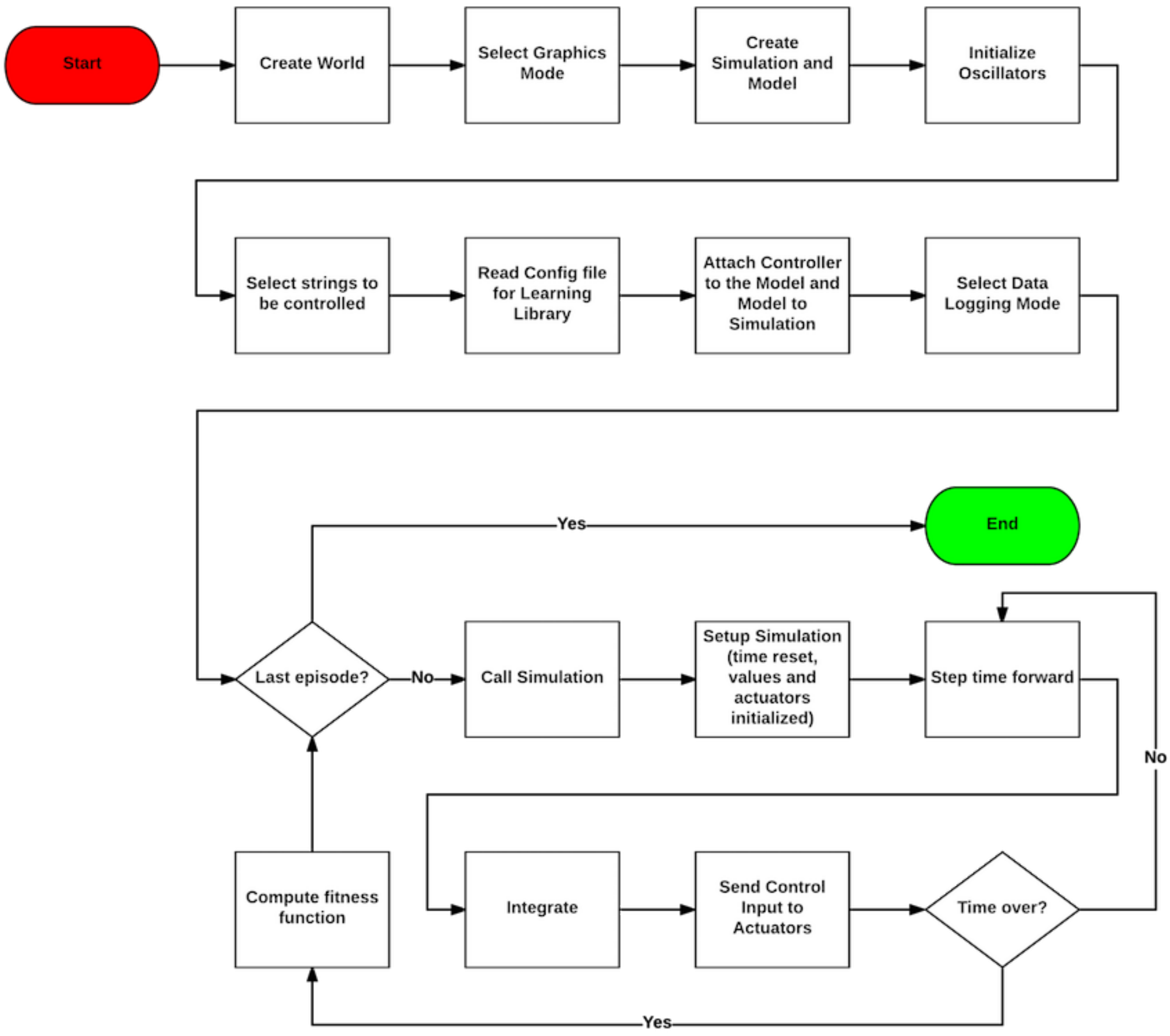

Start
Create World
Select Graphics Mode
Create Simulation and Model
Initialize Oscillators
Select strings to be controlled
Read Config file for Learning Library
Attach Controller to the Model and Model to Simulation
Select Data Logging Mode
Yes
End
Last episode?
No
Call Simulation
Setup Simulation (time reset, values and actuators initialized)
Step time forward
No
Compute fitness function
Integrate
Send Control Input to Actuators
Time over?
Yes

Fig. 4.9: Process flow of an NTRT app using the Learning Library's functions

A configuration file (describing for example number of open parameters in the search-space) and

[1]http://ntrtsim.readthedocs.io/en/latest/learning-library-walkthrough.html

the fitness function are the main additions in this case. Multiple calls to the simulations must also be added, as evolutionary algorithms require multiple evaluations in order to efficiently minimize the fitness function, thus yielding the best parameters values.

4.6.1 NTRT determinism issues

NTRT was introduced in section 2.3 as the open-source simulator used in tensegrity research. It is developed and used not only by NASA, but also by academic institutions such as Berkeley and Rutgers University[2].

However, as it is a constantly updated collection of open-source software modules, support for it is not as thorough as it would be for a well-established, paid software. Still, NTRT being developed exclusively for dynamic tensegrity structures, it has numerous advantages and particularities that allow easy implementation of tensegrity robots. This is the reason it is the simulator for tensegrities used within NASA.

There are two high-level methods of using NTRT: whether with or without graphical output. Without graphical output is particularly useful for running multiple episodes of a given simulation. However, it was discovered that there was a software bug that was causing determinism issues within the simulator. This means that setting the exact same controller parameters and initial states would not generate the exact same results after a given set time of simulation, as it should be expected.

For example, the NTRT app could be called with a YAML file. It would then say that SUPERball has moved a distance of y_1. Then, if a reset of the app is called, theoretically all initial states are the same, and SUPERball would move a distance of y_1 again. However, it moved in a different path and for a distance $y_2 \neq y_1$. This continued with all reset calls, i.e. calling it once more would result in a distance $y_3 \neq y_2 \neq y_1$, then $y_4 \neq y_3 \neq y_2 \neq y_1$, etc. An illustration of this can be seen in Figure 4.10.

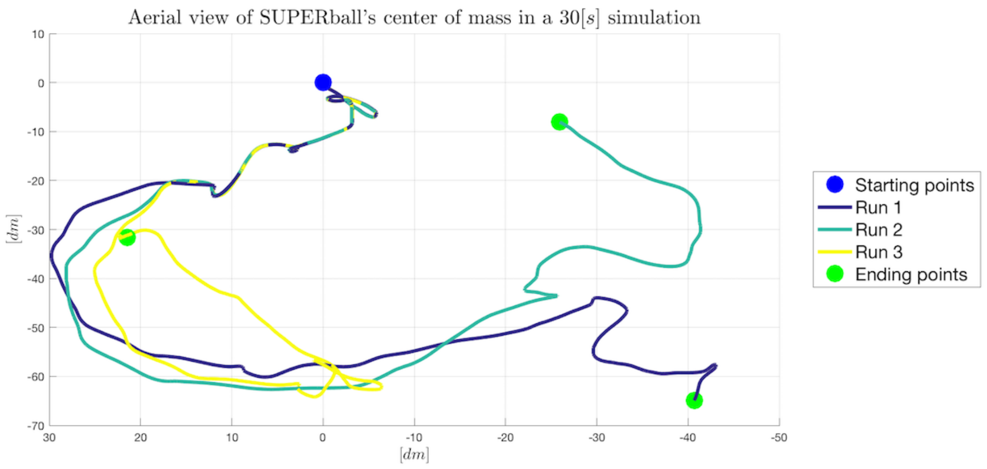


Fig. 4.10: Output of a simulation calling NTRT's reset function twice. SUPERball starts the simulations at the blue dots and the controller then runs from $t = 3[s]$ to $t = 30[s]$. After these $30[s]$, SUPERball is in one of the green dots and the controller is deactivated. The reset function is then called and it can be seen that the generated path is completely different from that of the 1st run. Similarly, the 3rd run is different from the 2nd and the 1st

[2]https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/graphs/contributors

This was not discovered earlier as the Learning Library functionalities had not yet been used by many people. A commit to another portion of NTRT's GitHub might thus have deprecated the base reset functions.

Moreover, by the time the author of this report was carrying out his project, he was the only one within the DTRL at NASA ARC using NTRT, and the main NTRT developers were not available to investigating these issues. A large portion of time was required to attempt to debug NTRT.

Firstly, it was verified that the errors were not coming from the controller's implementation. All random values or perturbations were removed, such that all states of the controller were the same. Alas at every run a different overall result was obtained. Memory leaks were also verified using Valgrind, and there were indeed none that were causing the non-deterministic behavior.

Next, it was verified that the issue was not coming from the implemented YAML model. To do so, the controller was deactivated and tensions were checked. Indeed, some vibrations at the start of a simulation could also be the error's source, as it can be seen in Figure 4.11.

Pretensions were thus adapted in the YAML model to make sure no initial vibrations were present in the simulation, as it can be seen in Figure 4.12 and Figure 4.13. Still, for practical reasons it will be easier if this issue proves not to be the issue, as in the physical prototype of the robot, it will be easier to control all actuators if they are all calibrated to be under the same pretension.

In addition to it, the ground on NTRT is built out of two parallel planes, the lower one corresponding to a height of $0[m]$. If part of the structure is between both of them at the start of the simulation, this may cause other problems as the simulator would try to quickly push it through the upper solid plane, thus generating very high tensions. The SUPERball model was thus adapted to make sure that no nodes were between the ground planes nor directly in contact with either of them. It was therefore dispatched millimiters above the upper ground plane, which after a first impact on the ground should not create very different tensions on the robot's cables.

Finally, even different numerical integration methods were checked. Neither Euler integration nor using the C++ Boost library (which can integrate using a 4th order Runge-Kutta method[3]) were reproducing the same result upon calling the NTRT reset functions.

An ongoing collaboration with the other NTRT developers is underway to try to get deterministic behavior out of the simulator. Specifically, a preliminary patch was added that modified how Bullet's colliding objects interact. This did not correct the reset issues either, though it made the results obtained with or without graphical output closer to one another (up to floating-point operations precision).

[3]http://www.boost.org/doc/libs/1_56_0/libs/numeric/odeint/doc/html/boost_numeric_odeint/getting_started/overview.html

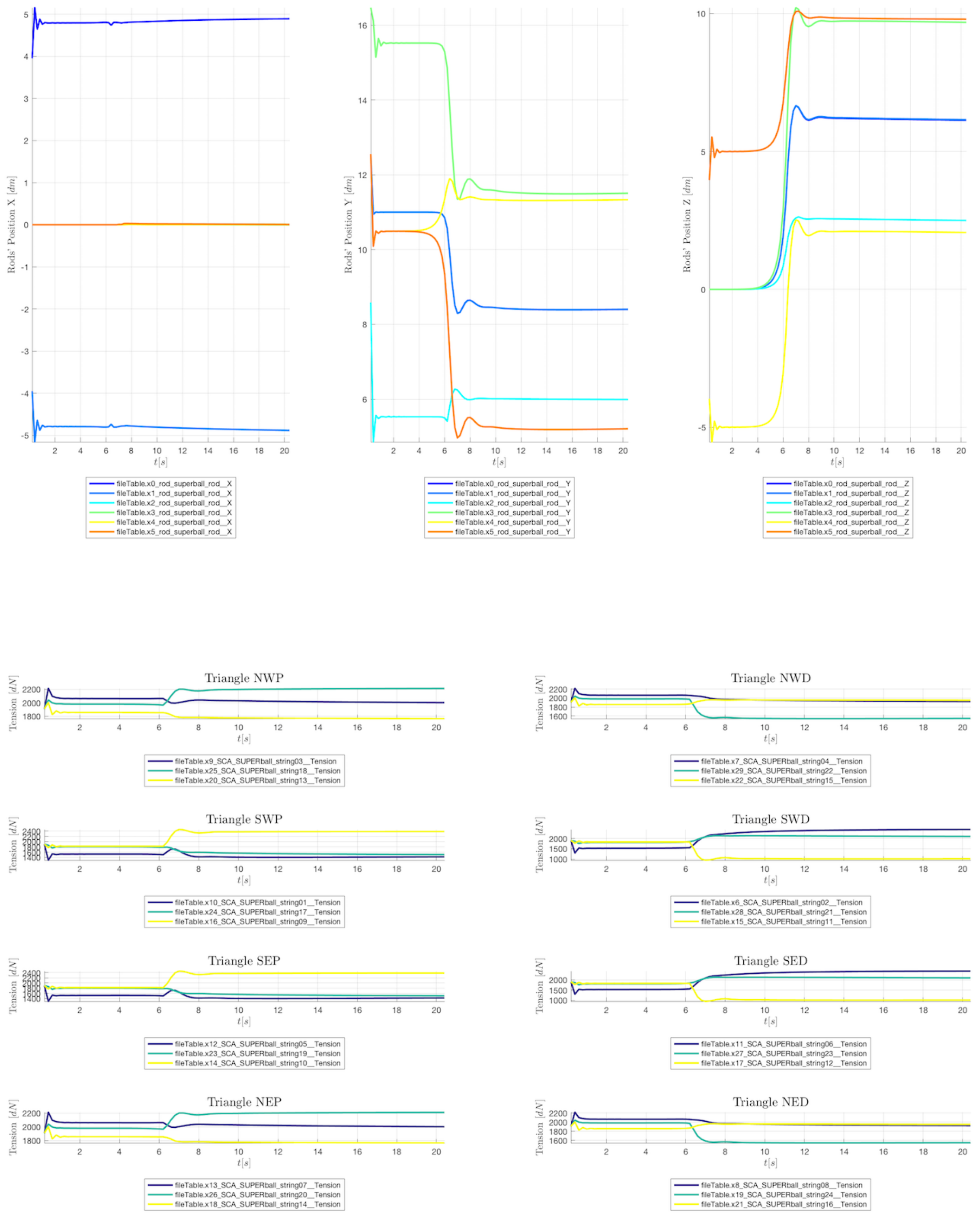


Fig. 4.11: Logged data showing that the initial YAML model provoked some vibrations in the rods' positions (top) and the cables' tensions (bottom)

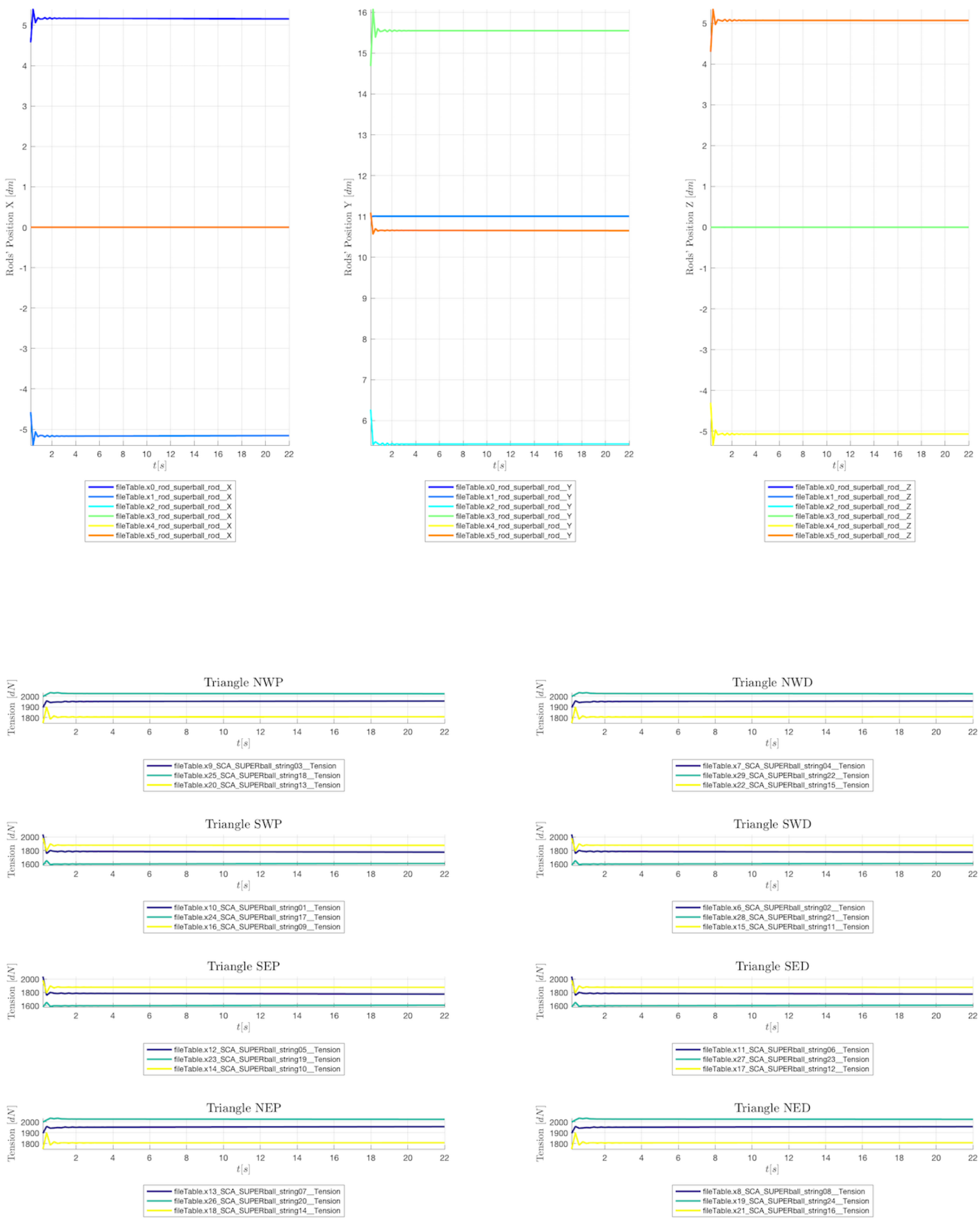
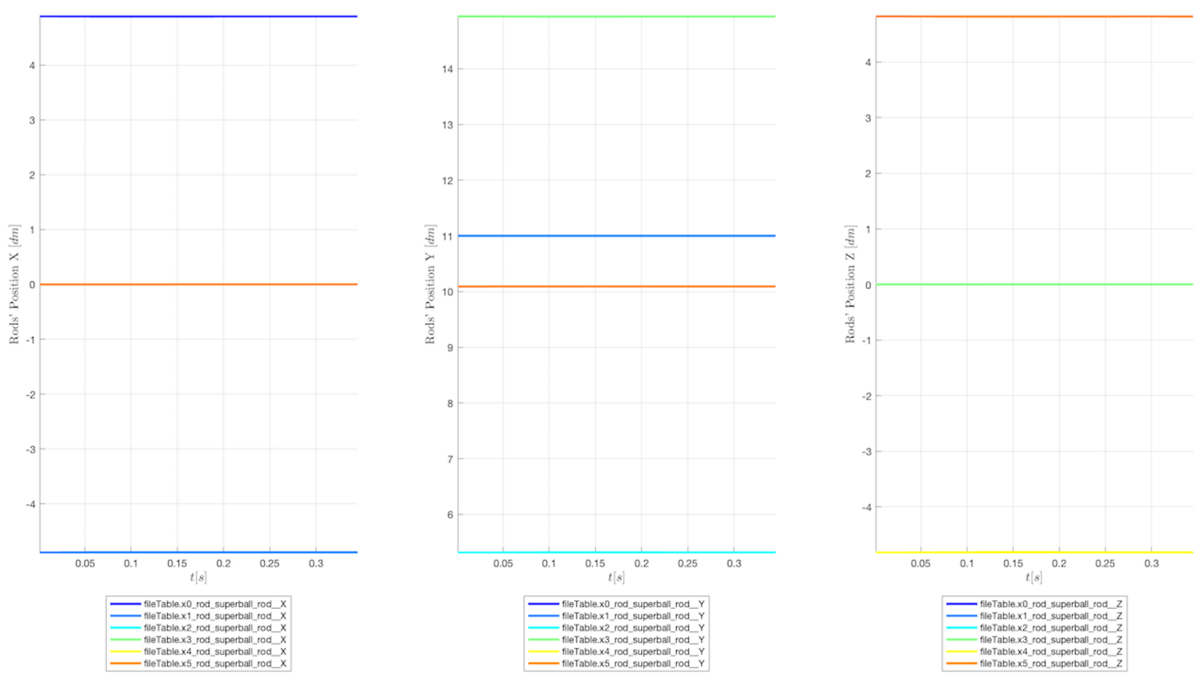


Fig. 4.12: Intermediate calibration of the YAML model to reduce initial vibrations. All vibrations on the rods (top) and the cables (bottom) are completely attenuated after $6[s]$



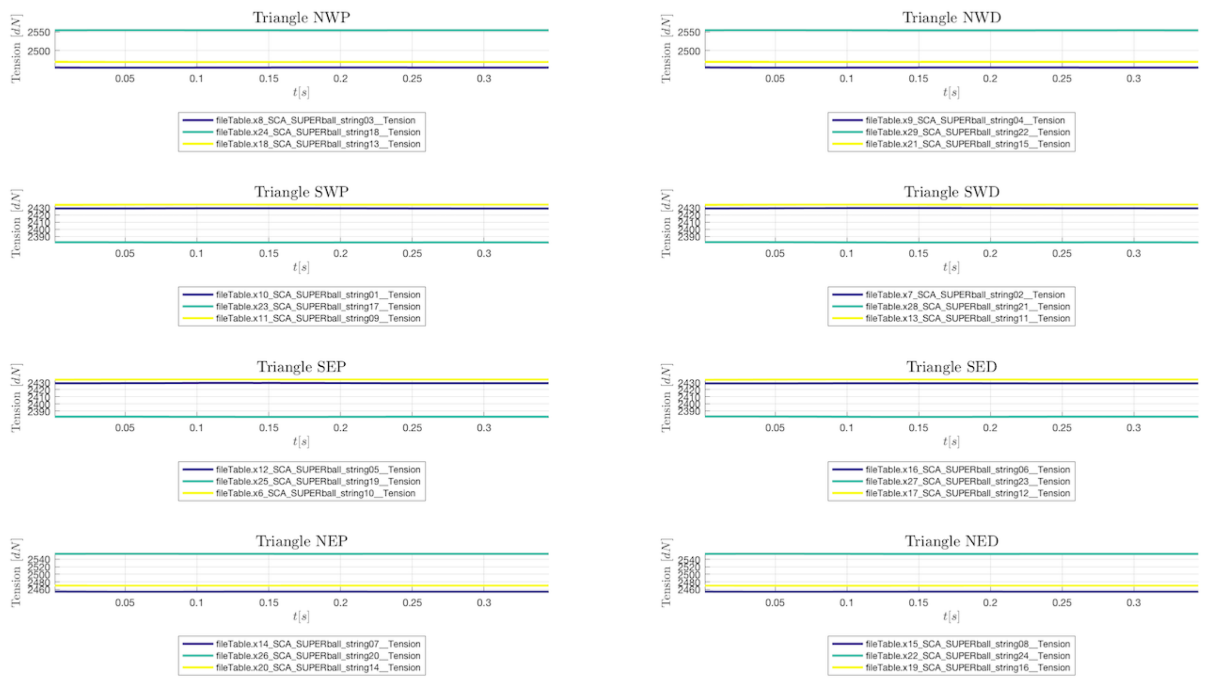


Fig. 4.13: Vibration-reducing calibration of the YAML model completed. There are no vibrations whatsoever on the rods (top) nor the cables (bottom) when the SUPERball model is dispatched on the NTRT app

Chapter 5

Results

5.1 Parameter search for double chain of oscillators model

5.1.1 Selected optimization algorithm

With the patch resolving part of the determinism issues within NTRT, it was then possible to try to find values for SUPERball's double chain of oscillators model introduced in subsection 4.5.2. Nevertheless, trying to tune the 16 open parameters of the coupling matrix **A** in Equation 4.3 by hand would simply be impossible, due to the highly non-linear and coupled nature of this model. What's more, four additional open parameters would also be added to the model, namely amplitude factors α_i, as the commands being sent to the strings are given in Equation 5.1. There are only four and not eight as it was assumed that that simplification was possible thanks to the symmetry of the double chain of oscillators model ($\alpha_i = \alpha_j$, $\forall i = 1, 3, 5, 7$ and $j = i + 1$).

$$l_i(t) = l_{i_0} \cdot (1 + \alpha_i \cdot \cos(\phi_i(t) + \phi_{i_0})) \tag{5.1}$$

This is the reason why it was first desired to use NTRT's Learning Library. However, with the reset functions not being fully resolved, it was necessary to find another solution to test this algorithm. Seeing that NTRT is called from a Linux terminal command line along with the YAML model as an argument, a Python script was implemented to run an evolutionary algorithm. There is indeed the `pyswarm` library[1], which is capable of running the Particle Swarm Optimization (PSO) algorithm, whose process flow can be found in Figure 5.1.

This algorithm was selected as the author of this report was already familiar with it, and because it has also already been used for achieving robust CPG implementations, such as in [20]. However, in order to run the optimization, a cost function needed to be designed.

A minimalistic cost function to be minimized was the selected approach to this problem. Indeed, it was simply set as the overall horizontal distance between the starting and final positions of SUPERball's center of mass. Consequently, the fact that the robot would try to follow a single

[1]http://pythonhosted.org/pyswarm/

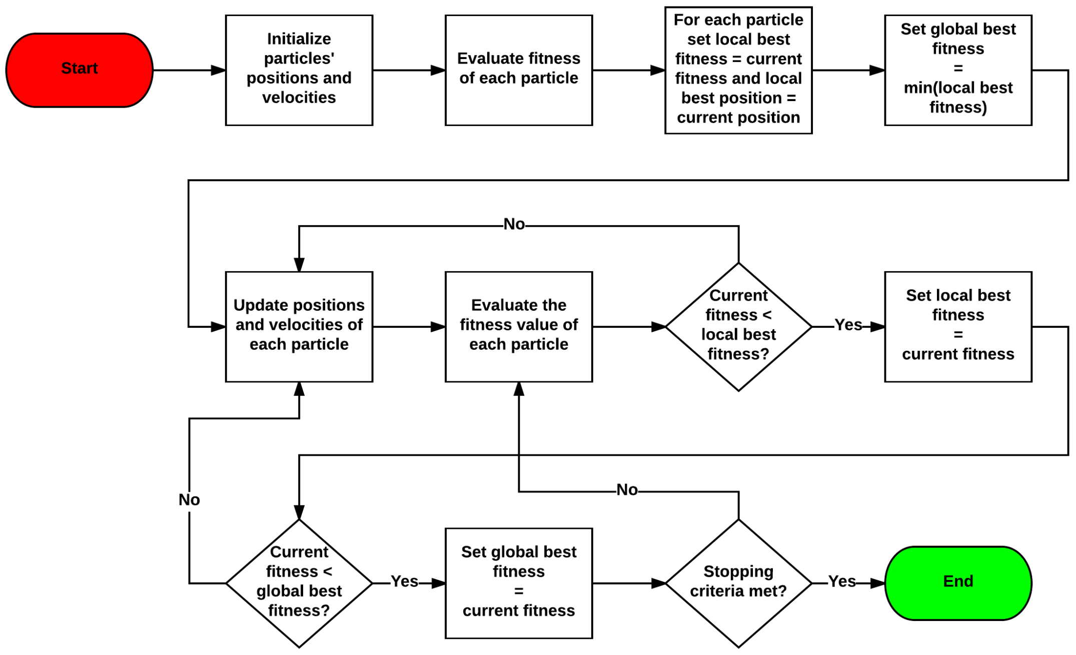

Start
Initialize particles' positions and velocities
Evaluate fitness of each particle
For each particle set local best fitness = current fitness and local best position = current position
Set global best fitness = min(local best fitness)
No
Update positions and velocities of each particle
Evaluate the fitness value of each particle
Current fitness < local best fitness?
Yes
Set local best fitness = current fitness
No
No
Current fitness < global best fitness?
Yes
Set global best fitness = current fitness
Stopping criteria met?
Yes
End

Fig. 5.1: Process Flow for a Particle Swarm Optimization

direction along its path is incorporated into the optimization. Thus, the cost function is given in Equation 5.2, where **r** corresponds to the position vector of the center of mass, subscripts x and y both horizontal components, and subscripts i and f meaning initial and final, respectively.

$$C(\mathbf{r}) = -\sqrt{(r_{x_f} - r_{x_i})^2 + (r_{y_f} - r_{y_i})^2} \tag{5.2}$$

5.1.2 Optimization on flat terrain

The best results for a non-modulated intrinsic frequency signal can be seen in Figure 5.2. These results were found when running the optimization algorithm on flat terrain. As it can be seen, the dynamics each oscillator on the left part of the chain is in antiphase with its respective neighbor on the right part of the chain. There is indeed a direct link between the oscillators' output and the cables' tension, seeing that the latter varies with the cables' lengths, i.e. Equation 5.1.

What was interesting to see was that with this model, even if it was optimized on flat terrain, SUPERball's gait could have almost been considered that of a legged animal. As the frames on Figure 5.3 illustrate, two of the rods served as legs, whereas the horizontal ones parallel to the direction of movement seemed to stabilize the whole gait (the bottom one to have a 3$^{\text{rd}}$ contact point with the ground and the top one for the lateral shifts), just like a tail would in animal locomotion.

This gives us some insight into Biology. To the best of the author's knowledge, no macro-organism found in Nature presents any type of spherical or rotating locomotion system [21]. These results could therefore reinforce the idea as to why animals have evolved to have legs. Even if physically the animal was developing some members that would allow rolling, structural

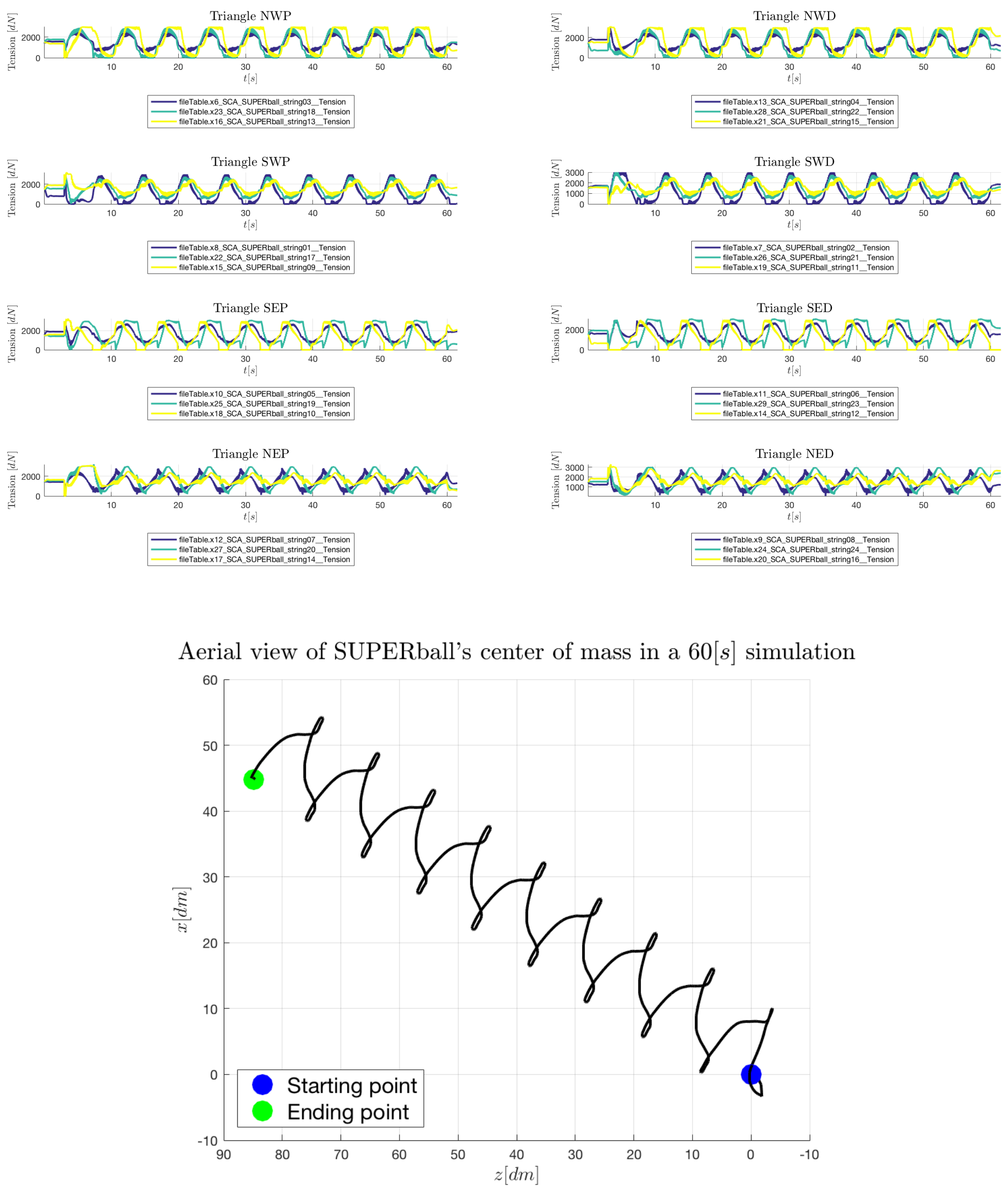
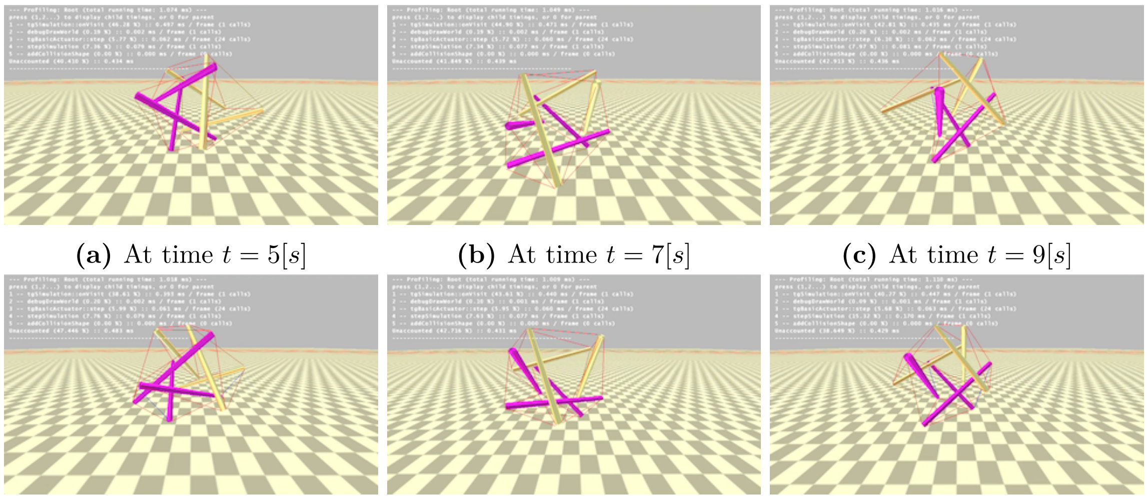


Fig. 5.2: Cables' tensions per triangle (top) and aerial view of the path done by SUPERball (bottom) after the best PSO on flat terrain

support structures give better overall stability, albeit they might require more energy. Consequently, natural selection promoted legs throughout the millions of years of evolution that have happened on Earth.

In order to showcase the smooth modulation that CPGs present, other tests using the parameters obtained with the PSO were carried out. Indeed, the intrinsic frequency ω_i may be changed online. The simulation results can thus be seen in Figure 5.4 and the video in `https://youtu.be/gph-_QV8AvQ`.



(a) At time $t = 5[s]$ **(b)** At time $t = 7[s]$ **(c)** At time $t = 9[s]$

(d) At time $t = 11[s]$ **(e)** At time $t = 13[s]$ **(f)** At time $t = 15[s]$

Fig. 5.3: A few frames of the best PSO results on flat terrain. The colors of each rod help showing that the obtained gait is not a rolling one, but surprisingly one where the main contact points with the ground are generally the same. For the full video, check https://youtu.be/uAiJJ9y0Iv4

5.1.3 Changing directions

In subsection 4.5.3, it was stated that the order of the oscillators on the chain could be permuted, as long as the neighbor constraints were kept. Theoretically this should allow SUPERball to change its direction of locomotion.

However, seeing that SUPERball is resting on the SWD triangle at the start of the simulation of Figure 5.3, this affected the possibilities of which gaits could be generated. In that initial optimization, the robot was moving Northwest, i.e. going over one of the cables. Additionally, the chain of oscillators was the one illustrated in Figure 4.7.

All other permutations were tested, and those that gave the best results are shown in Figure 5.5 and Figure 5.6.

The remapping of Figure 5.5 allowed SUPERball to go in a different direction with regards to the original mapping. Though it took longer for it to converge to a stable, periodic gait, the robot was still capable of maintaining its direction. The accompanying video can be found in https://youtu.be/Gi8B0cKJuSc.

The one on Figure 5.6 on the other hand took much longer to converge to a stable, periodic gait. The accompanying video can be found in https://youtu.be/-7o8L6szeLQ.

Due to time constraints, no further exploration of this steering mechanism was possible, but these preliminary results have already shown that it is possible to control (at least in part) SUPERball's direction with this controller.

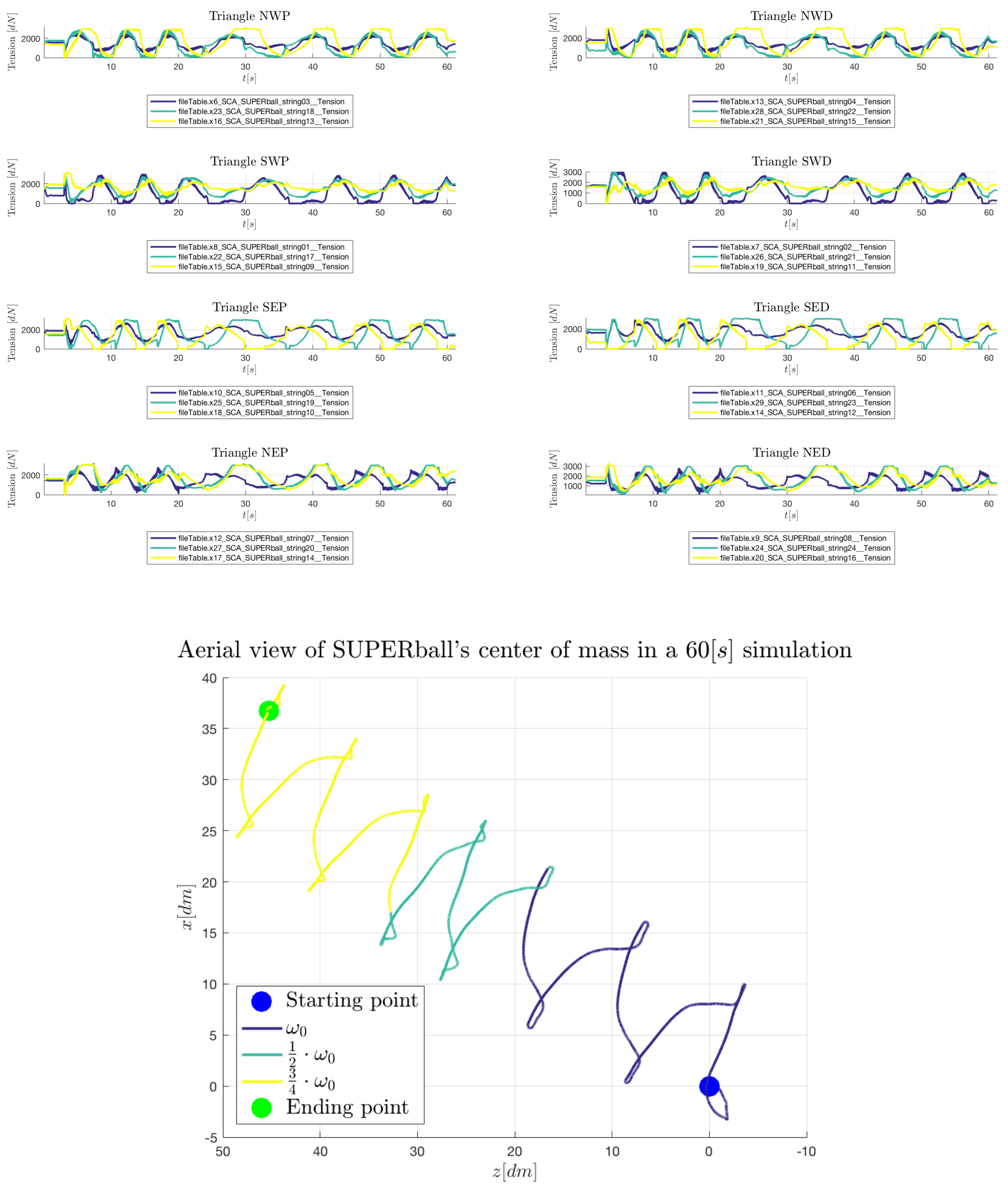


Fig. 5.4: Cables' tensions per triangle (top) and aerial view of the path done by SUPERball (bottom) after the best PSO on flat terrain with online frequency modulation. The simulation starts exactly like in Figure 5.2, but between 20[s] and 40[s] all intrinsic frequencies are half of the original ones, and between 40[s] and 60[s] they are all $^3/_4$ of the original ones

5.1.4 Optimization on uneven terrain

With this initial gait obtained, new tests could also be run to see if similar results would be obtained in uneven terrain. Instead of having a flat terrain, periodic ground oscillations could easily be implemented in NTRT (see Figure 5.7). Therefore, the ground was replaced and other PSOs were run. The best achieved results can be seen in Figure 5.8.

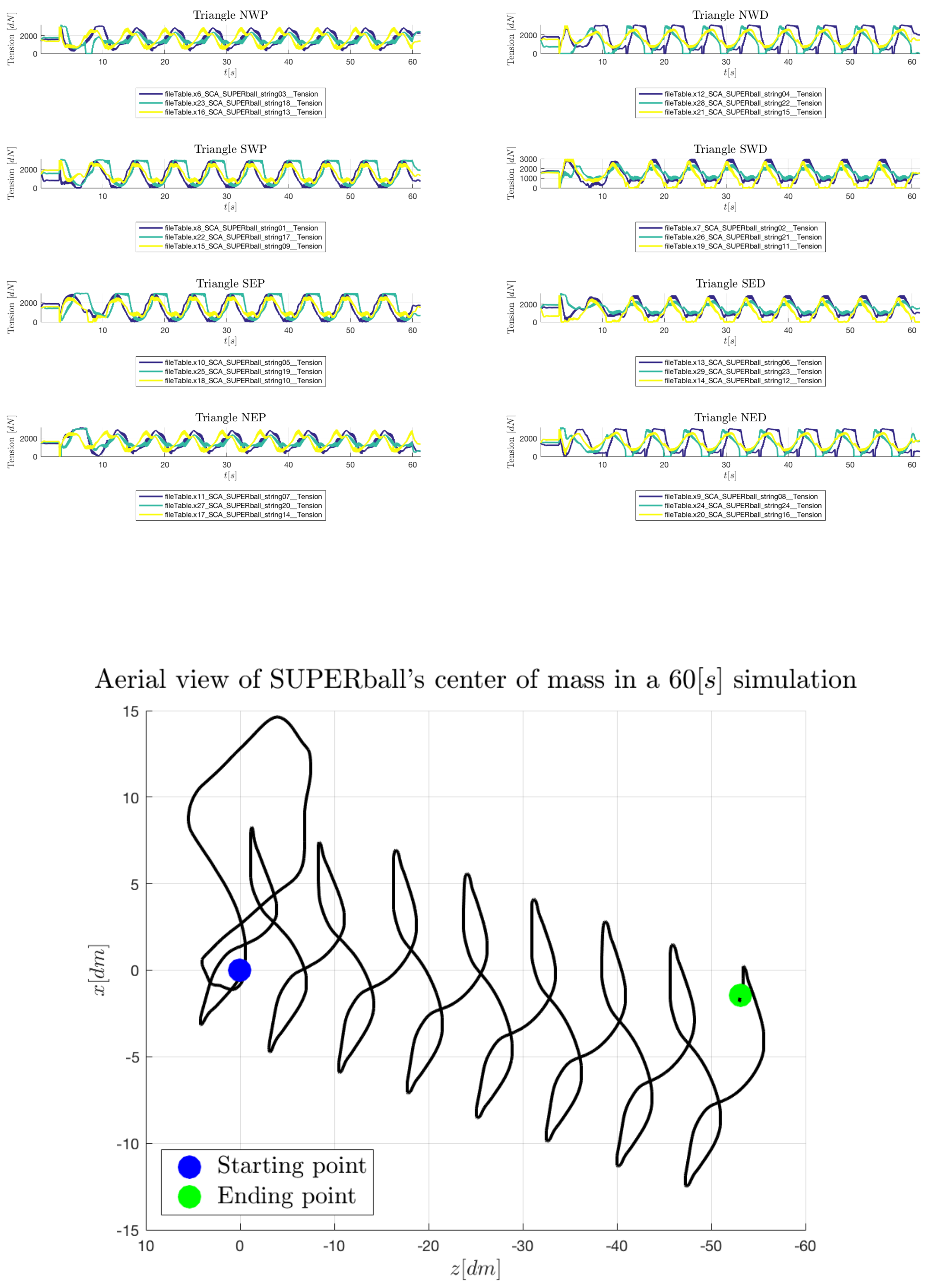
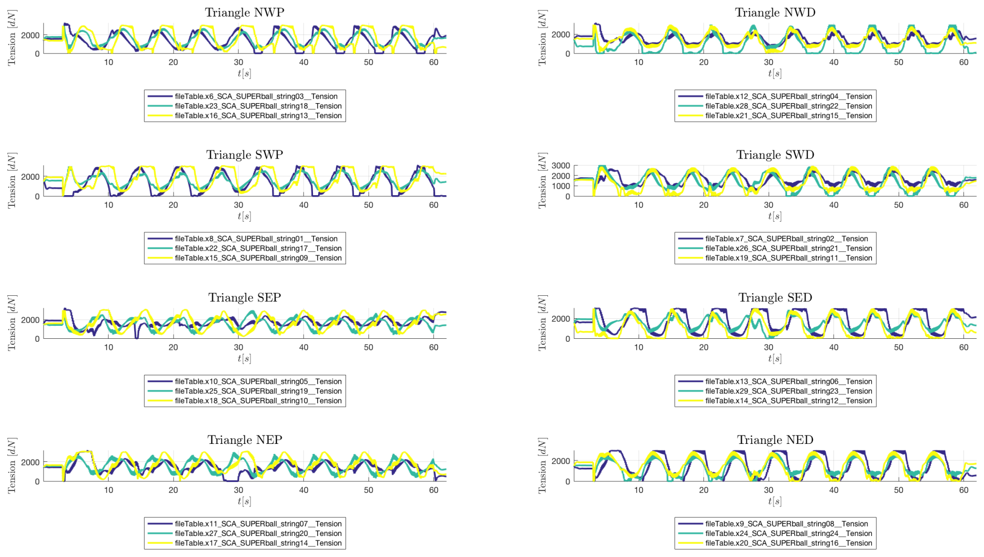


Fig. 5.5: Cables' tensions per triangle (top) and aerial view of the path done by SUPERball (bottom) after the best PSO on flat terrain. The double chain of oscillators was remapped to be SEP-SWP-SED-SWD-NED-NWD-NEP-NWP, i.e. that of Figure 4.8

As it could have been expected, moving on rough terrain is much harder than on flat one. Though the results do not generate a very periodic gait as in the flat ground tests, SUPERball did follow a general direction trend. A video of this simulation can be found in `https://youtu.be/bR_taQyOHdY`.

Finally, modulating the signals is also possible. The same modulation as the flat ground test was done and though it traveled a smaller distance, it still kept moving into a general direction, as it can be seen in Figure 5.9. Its respective video is located at `https://youtu.be/f8bUB65O74g`.



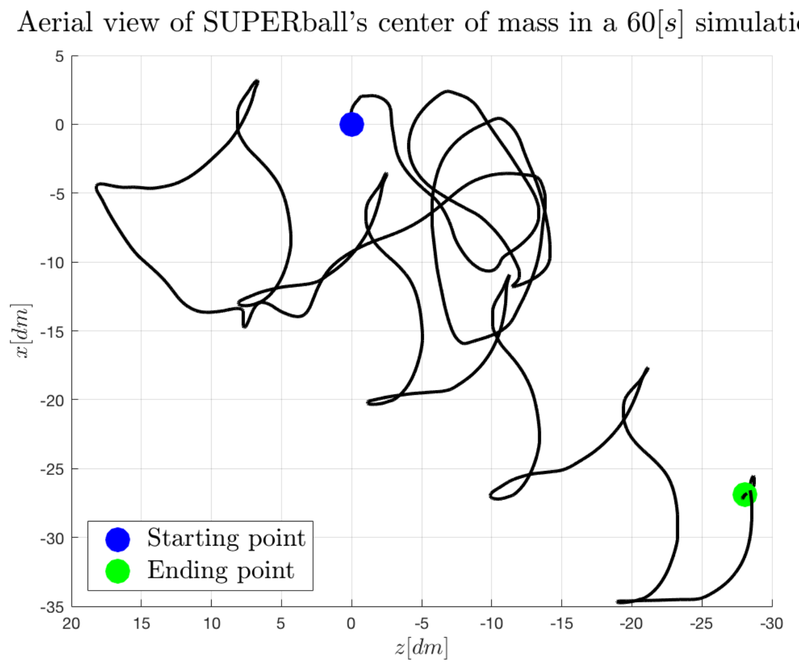
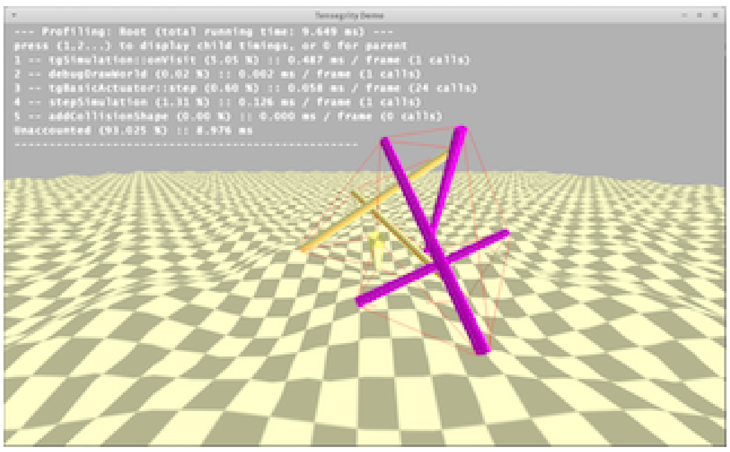


Fig. 5.6: Cables' tensions per triangle (top) and aerial view of the path done by SUPERball (bottom) after the best PSO on flat terrain. The double chain of oscillators was remapped to be NED-SED-NWD-SWD-NWP-SWP-NEP-SEP



```
--- Profiling: Root (total running time: 9.649 ms) ---
press (1,2...) to display child timings, or 0 for parent
1 -- tgSimulation::onVisit (5.05 %) :: 0.487 ms / frame (1 calls)
2 -- debugDrawWorld (0.02 %) :: 0.002 ms / frame (1 calls)
3 -- tgBasicActuator::step (0.60 %) :: 0.058 ms / frame (24 calls)
4 -- stepSimulation (1.31 %) :: 0.126 ms / frame (1 calls)
5 -- addCollisionShape (0.00 %) :: 0.000 ms / frame (0 calls)
Unaccounted (93.025 %) :: 8.976 ms
-------------------------------------------------
```

Fig. 5.7: Example of uneven terrain built with NTRT's `tgHillyGround` class, with a triangle size factor of 4 and a wave size factor of 2

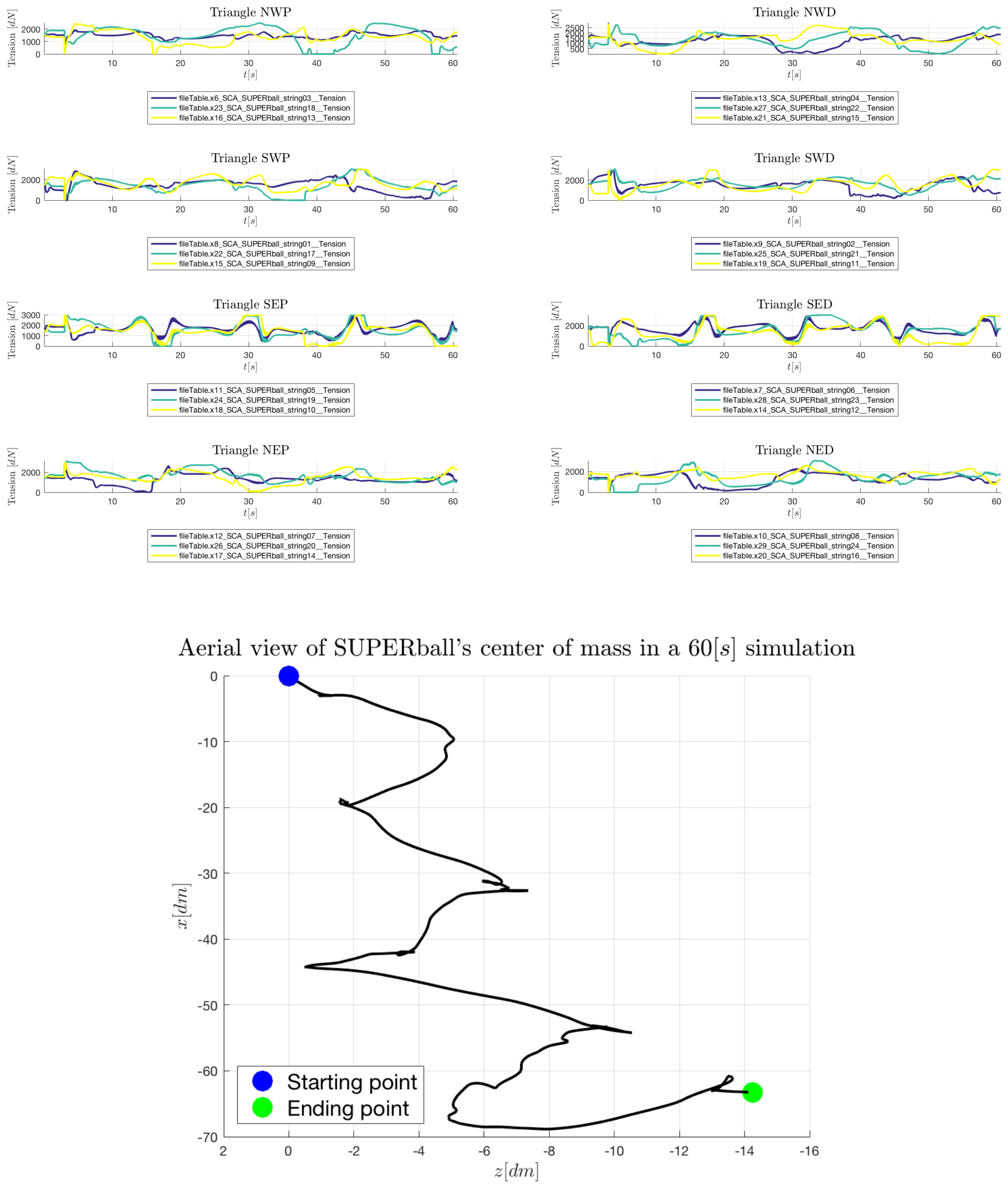


Fig. 5.8: Cables' tensions per triangle (top) and aerial view of the path done by SUPERball (bottom) after the best PSO on the terrain shown in Figure 5.7

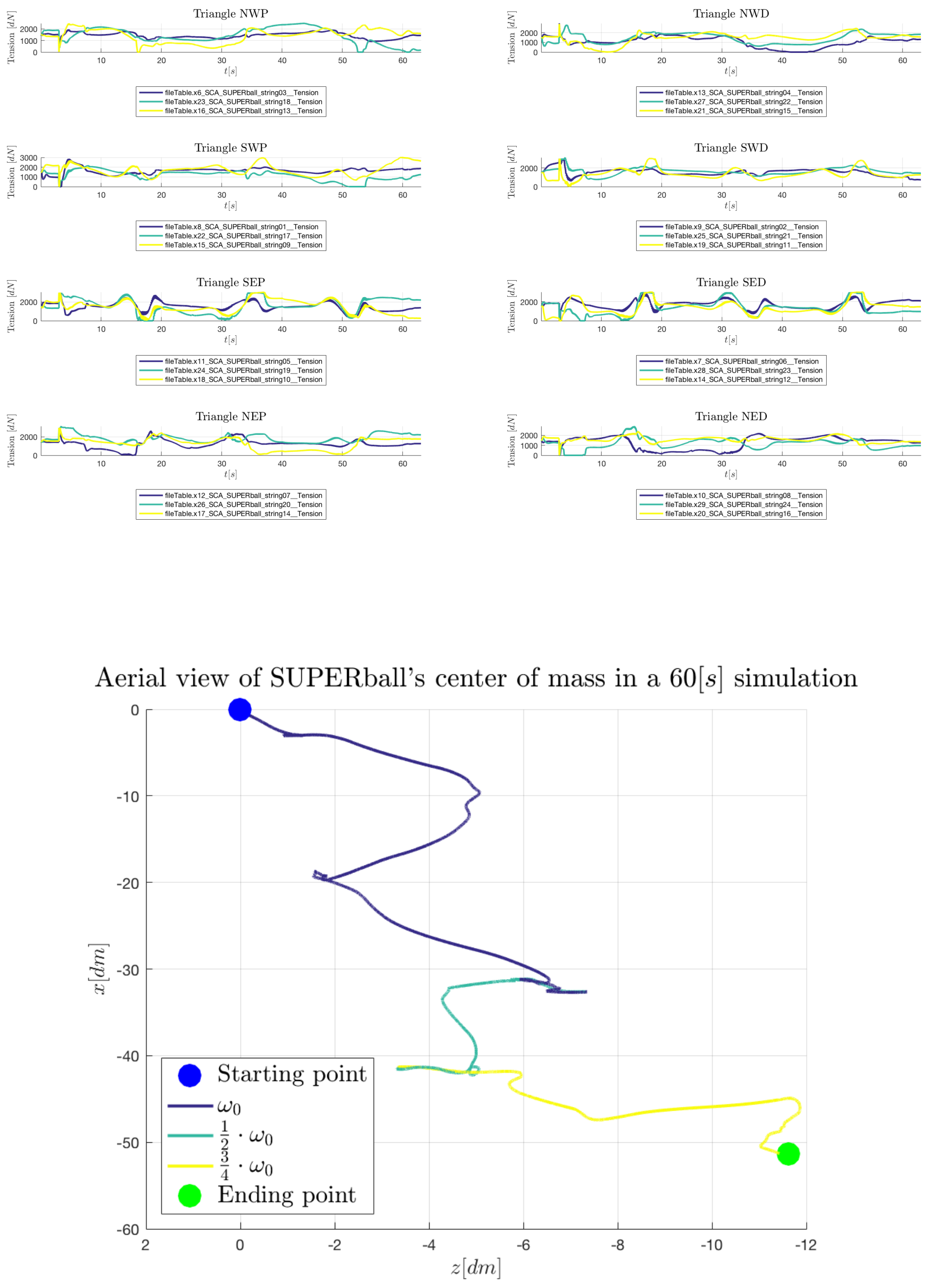


Fig. 5.9: Cables' tensions per triangle (top) and aerial view of the path done by SUPERball (bottom) after the best PSO on the terrain shown in Figure 5.7

5.2 SUPERball v1.0 Motor Primitives

The issues with NTRT prompted the need to take a different approach as to how CPGs should be incorporated with SUPERball's locomotion. The first version of the robot exhibited simple forward locomotion. This was achieved thanks to an offline optimization method using Reinforcement Learning, specifically a modified version of the Guided Policy Search (GPS) algorithm named Mirror Descent Guided Policy Search (MDGPS).

The GPS algorithm can be seen as a trajectory-optimized direct policy search scaled to high-dimensional systems that avoid local minima [22]. The MDGPS one optimizes the global policy by sampling the current iteration's local policies and approximating the minimum divergence between the global and local policies [19].

For that publication, SUPERball v1.0 learned a rolling gait using transitions between the robot's "faces" as the local policies and how to transition between them with limited sensor data as the global data. The robot having only 12 actuators, the expected actuation pattern and how that would be translated into forward locomotion can be seen in Figure 5.10. Subsequently, those actuators have been fed into the MDGPS algorithm and created a signal that would allow SUPERball v1.0 to make one full roll, i.e. return to the same equilateral triangle touching the ground.

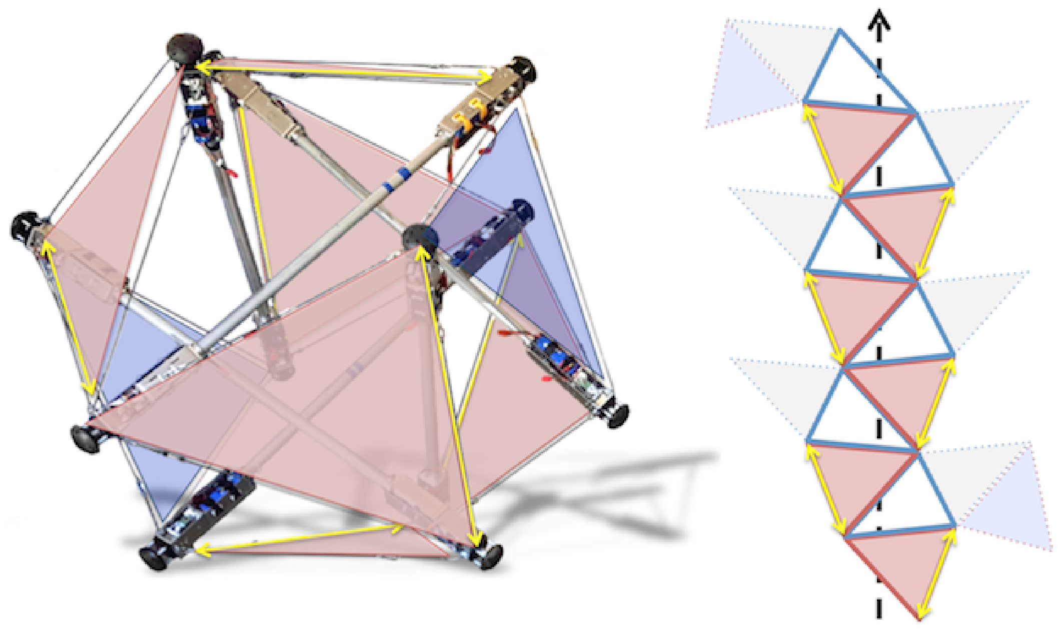


Fig. 5.10: Actuation pattern used on SUPERball v1.0. There are 8 equilateral triangles, shown in either red or blue. Each red triangle represents a face with only one cable actuated. These actuated cables are denoted by the yellow double arrows. Each blue triangle represents a face with actuators on all cables. The basic forward rolling of SUPERball has the robot landing with red triangles on the ground during locomotion. The left figure shows each triangle highlighted on SUPERball and the right one shows the forward locomotion pattern, or walking pattern, of SUPERball, taken from [19]

These motor primitives have been validated with the robot's first physical prototype. However, seeing that the GPS algorithm requires a large amount of computational power, it would not be an effective long-term solution to make SUPERball have robust locomotion. A more online approach should be taken, and this is where the CPGs come in. If the CPGs can reproduce these motor primitives, as well as modulate them and have smooth transitions between different frequency regimes, then this could be a better solution to be explored.

The discrete signals that allowed SUPERball v1.0 can be seen in Figure 5.11. It can also be seen that they do not seem to be periodic by nature. However, seeing that these $30[s]$ signals make the robot complete one full roll, if the signal can be reproduced continuously then theoretically the robot should achieve continuous rolling.

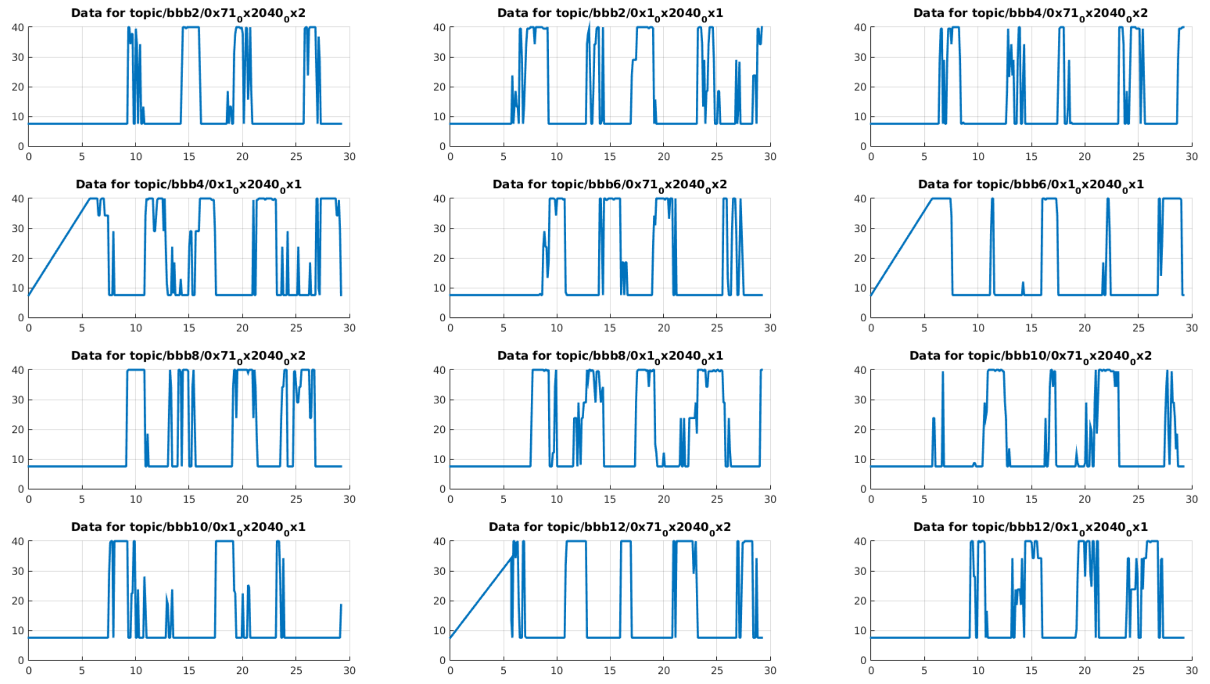


Fig. 5.11: Signals obtained with the GPS algorithm for SUPERball v1.0

In order to create a continuous periodic function, Fourier series have been used to interpolate the GPS results. Indeed, given a signal $s(x)$ with variable x, it is possible to represent it as a partial or infinite sum of harmonically related sinusoidal functions, thanks to the frequency ω. The Fourier series equation is given in Equation 5.3.

$$s_N(x) = \frac{A_0}{2} + \sum_{n=1}^{N} \left(a_n \cdot \cos(n \cdot \omega \cdot x) + b_n \cdot \sin(n \cdot \omega \cdot x) \right) \tag{5.3}$$

However, using MATLAB version R2016b only allows a limited number of sinusoidal functions, specifically eight. These 8-component interpolations can be seen in Figure 5.12.

Nevertheless, each obtained signal has a different main frequency. To simplify the CPG controller, each signal will be cut to at the period of the smallest period, i.e. $21.14[s]$ according to Figure 5.12. What's more, hardware constraints on the physical prototype were required for safety reasons. This also has to be incorporated into the interpolated signals.

The resulting motor primitives, alongside the early data can be seen in Figure 5.13.

These new motor primitives can then be fed to the amplitude-controlled CPGs described in subsection 3.2.2 so that frequency may easily be modulated. For verification purposes, a non-varying driving signal was applied such that the signals from Figure 5.13 are reproduced. This test can be seen in Figure 5.14, where everything was scaled such that the motor primitives were reproduced if $d = {}^{1}\!/_{30}$.

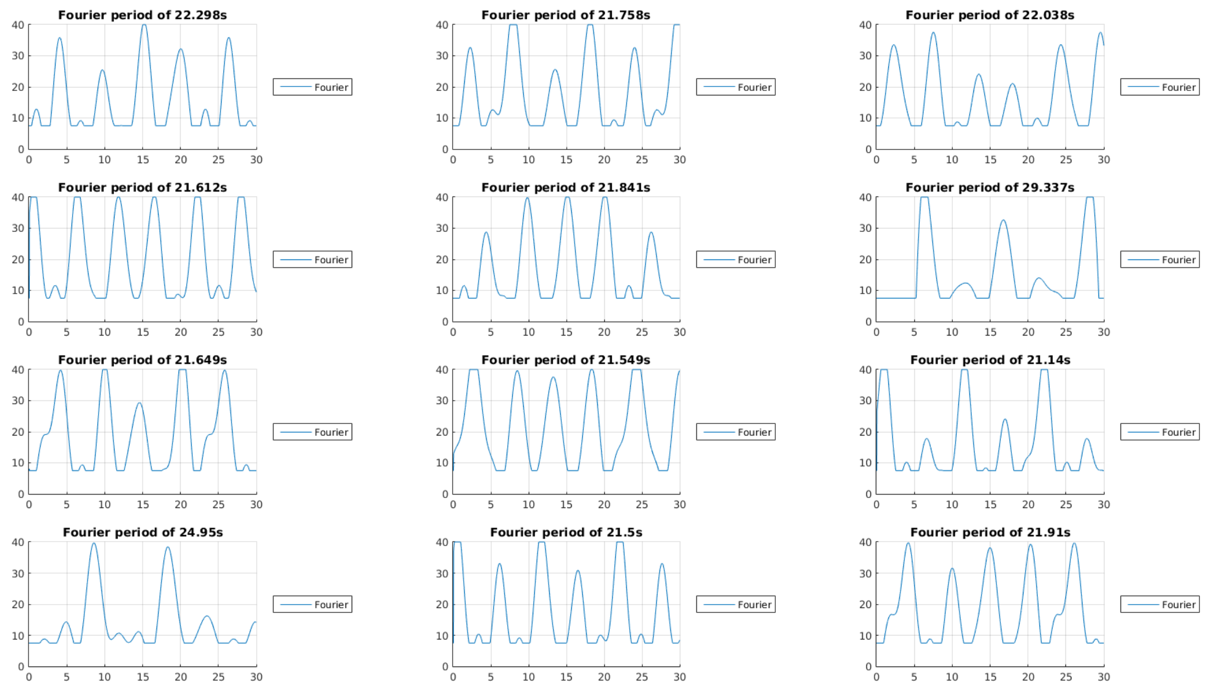


Fig. 5.12: 8-component Fourier series interpolation of the signals in Figure 5.11

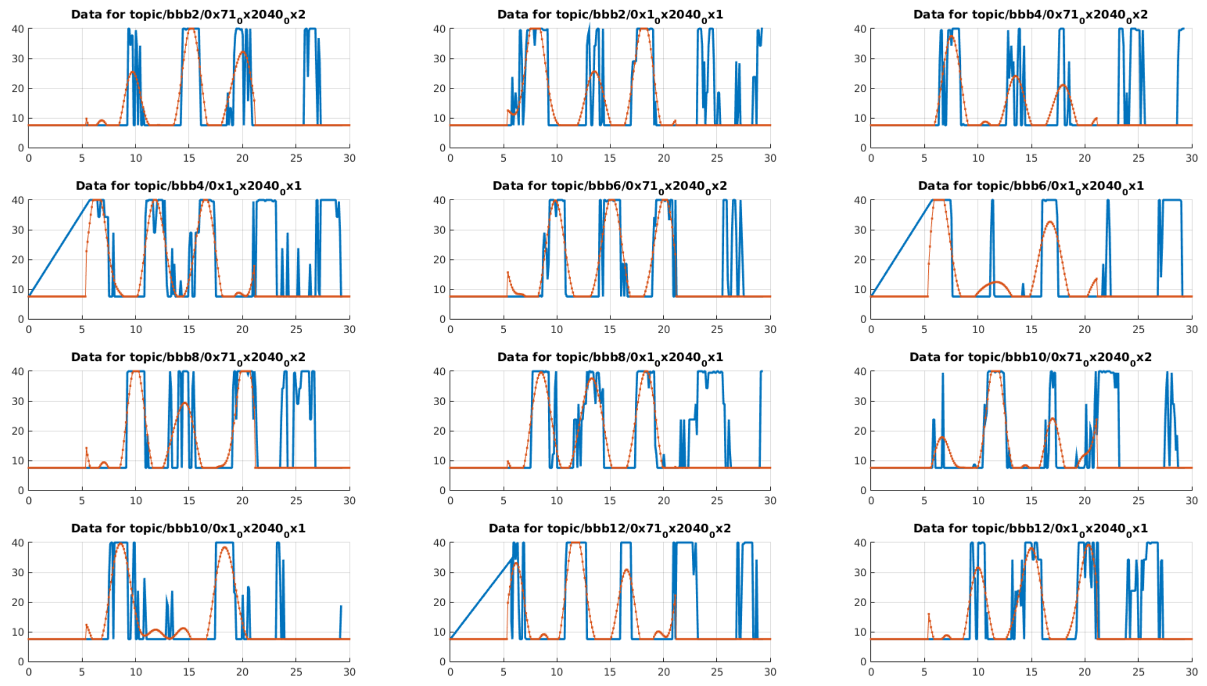


Fig. 5.13: GPS signals alongside thresholded Fourier series interpolation

The results of this can be seen in Figure 5.15. The first and last selected frames have the robot resting on the same base triangle, thus indicating that a full roll was obtained (the full video can be found in `https://youtu.be/GX8vK5J-RHQ`). Removing part of the results from the signals to be learned was therefore not harmful. The robot is still capable of rolling.

The signal reproduction being successful, it is now possible to try to modulate the frequencies, as it can be seen in Figure 5.16 (with a linearly varying drive signal), and in Figure 5.17 (with a single modification of the drive signal).

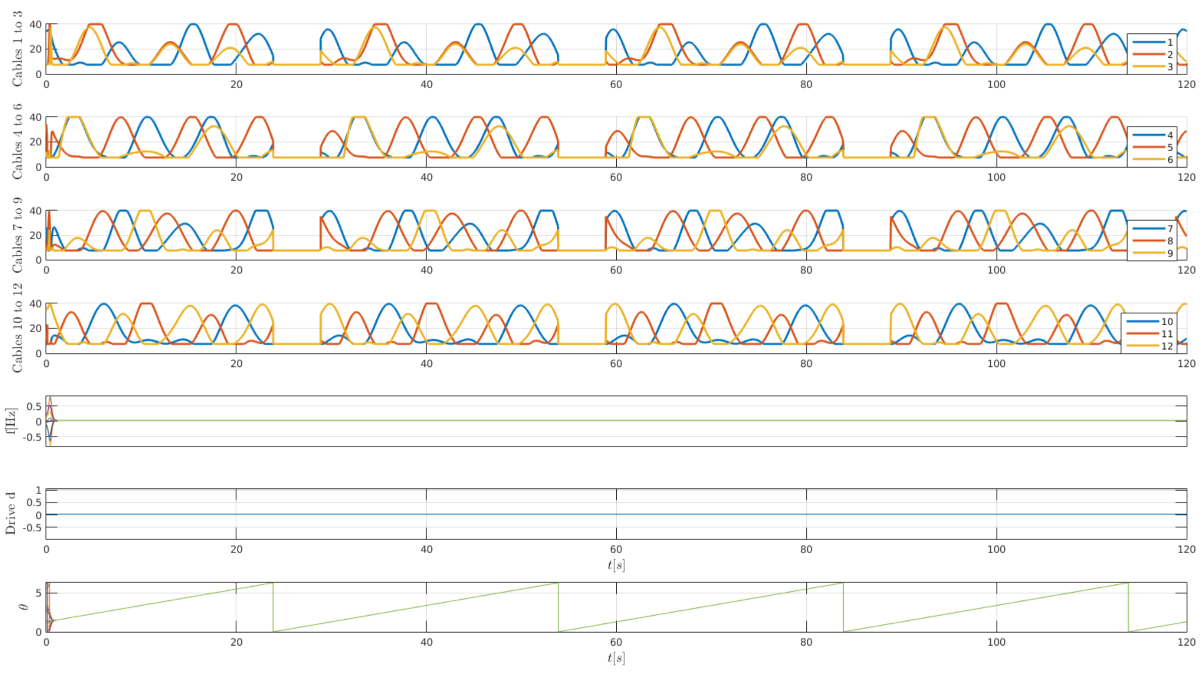
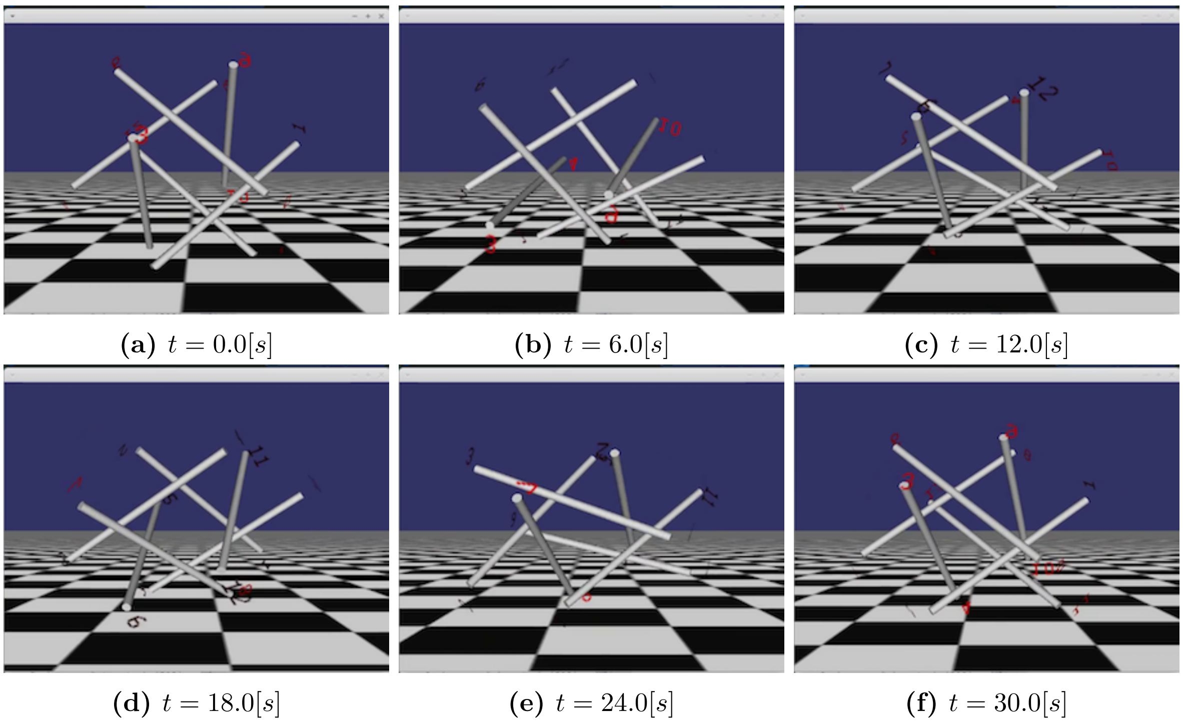


Fig. 5.14: CPG signals constructed with the Fourier series interpolation



(a) $t = 0.0[s]$ **(b)** $t = 6.0[s]$ **(c)** $t = 12.0[s]$
(d) $t = 18.0[s]$ **(e)** $t = 24.0[s]$ **(f)** $t = 30.0[s]$

Fig. 5.15: Results obtained with the CPG controller learning a signal similar to the one used on SUPERball v1.0

The accompanying video to Figure 5.16 (`https://youtu.be/IhnE1WFxQGY`) shows that SUPERball was unable to roll after about 40[s]. Seeing that the drive was increasing from $d = 1/30$ to $d = 1/6$ in 120[s], it can be deduced that the maximum speed with these signals on flat terrain is approximately 2.3 times the original one.

This test was done to illustrate the smooth transition that happens even when a step function is used to change the oscillators' frequencies. It was indeed doubled at $t = 60[s]$. The accompanying video is found in `https://youtu.be/TnAuZvZVzfM`.

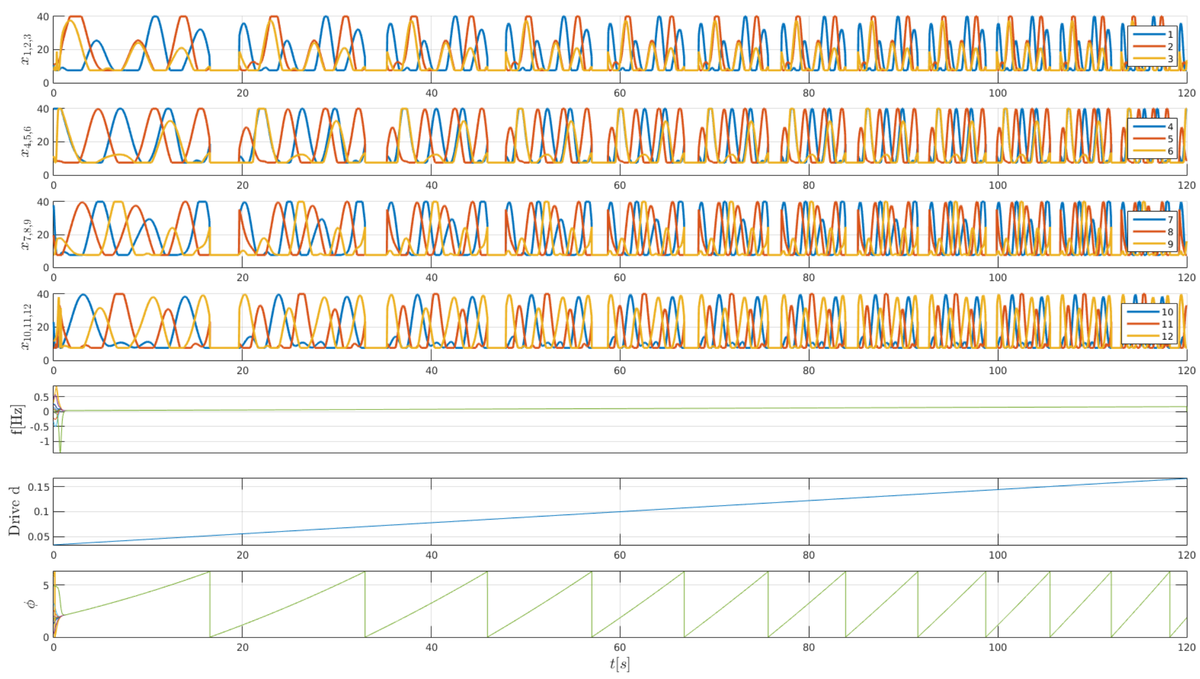


Fig. 5.16: CPG signals with linearly increasing amplitude drive modulation

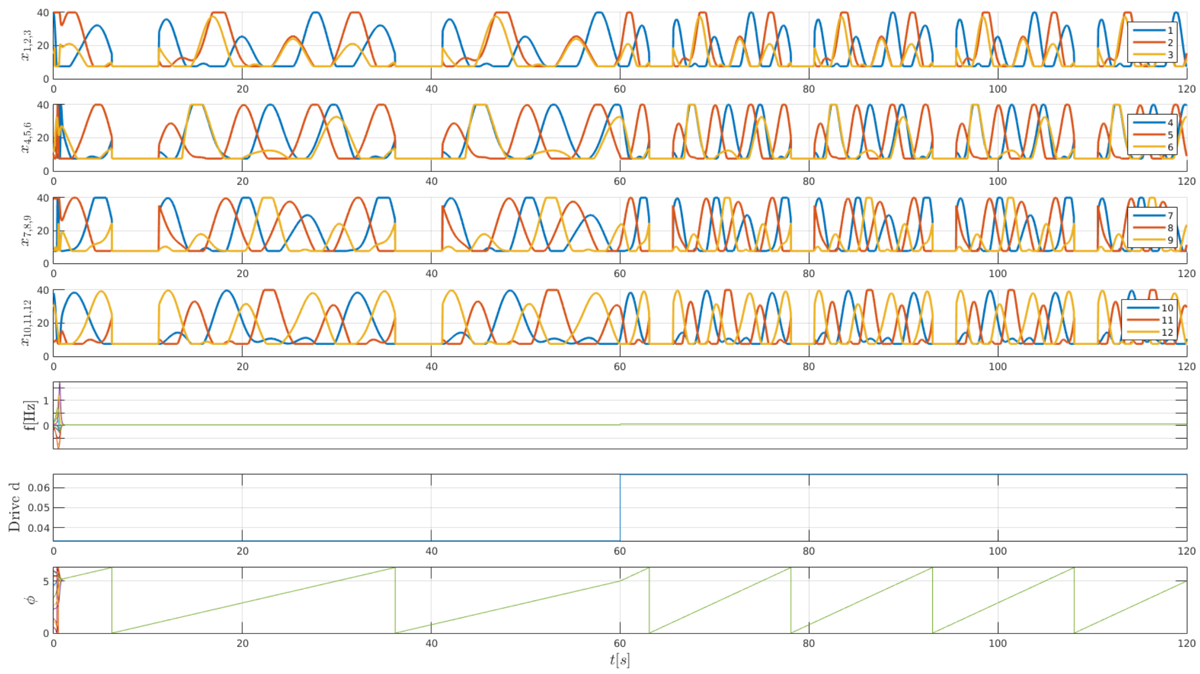


Fig. 5.17: CPG signals with step amplitude drive modulation

All the results of this section were obtained using OpenSceneGraph (OSG) as the 3D rendering program for NTRT (instead of GLUT/FreeGLUT) as the GPS framework was installed in a computer that had this former configuration of NTRT. What is interesting to know though, is that in that computer there was also a Robot Operating System (ROS) interface setup, which would easily allow the transition from simulation to physical testing.

With this preliminary framework working, it was expected to rerun the offline GPS optimization, as well as collaborate with Rutgers University, who have been working not only on GPS, but also on a kinodynamic planner to generate new teaching data for a 24-actuator version of a tensegrity icosahedra, i.e. similar to SUPERball v2.0. However, software bugs prevented this new data to be obtained.

Due to time constraints this implementation has thus been put on hold, though the whole framework for generating the CPGs out of the motor primitives is functional and can be found in the report's author's personal repository in NTRT's GitHub page[2].

5.3 Matlab implementation

5.3.1 Incorporating Tegotae to SUPERball

The Tegotae approach first described in subsection 3.2.2 offers a method of incorporating feedback components into the CPGs through physical coupling instead of the neural coupling used in the other implementations.

SUPERball being a tensegrity robot, all sensed forces are distributed across the whole network of cables and rods. Thus, it would be interesting to see if taking inspiration from the methodologies obtained in [17] could be used to generate locomotion for SUPERball.

However, in that article, the oscillators were directly connected to the limbs, who were the elements sensing ground reaction forces (GRFs). There were indeed three-axis force sensors at each foot of their robot. In the case of SUPERball, it isn't the rods that are directly connected to the oscillators, but the cables. No GRFs can then be directly sensed. Still, as each actuator is directly connected to two end caps, those could serve as the interface between the physical environment and the actuators' phases. Indeed, one could imagine that if a single end cap perceives an impact, the cables connected to it shall suffer a change in tension, which is being controlled by the CPGs.

The matrix illustrated in Figure 4.3 (right) would then be incorporated into this report's Tegotae implementation. Moreover, as the tensegrity icosahedra is such an interconnected network of components, using only a single Tegotae rule surely wouldn't fully capture the whole robot's complexity. The other parts that could have a significant impact on a cable's tension are the opposite ends to which the actuators are connected to. Certainly, if one of the end caps suffers a strong GRF, it is safe to assume that the cables connected to its opposite end shall increase their actual lengths, thus increasing their tensions.

Subsequently, a Tegotae rule incorporating these dynamics shall also be used. The matrix given in Figure 5.18 illustrates these "neighboring" connections between an actuator and the "distant" end caps.

The established Tegotae-based CPG equation used for SUPERball is therefore Equation 5.4.

$$\dot{\phi}_i = \omega - \sigma_1 \cdot \left(\frac{1}{n_{L_1}} \cdot \sum_{j \in L_1(i)}^{n_{L_1}} N_j^V \right) \cdot \cos\phi_i + \sigma_2 \cdot \left(\frac{1}{n_{L_2}} \cdot \sum_{j \in L_2(i)}^{n_{L_2}} N_j^V \right) \cdot \cos\phi_i \tag{5.4}$$

[2]https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/tree/SUPERball_CPG

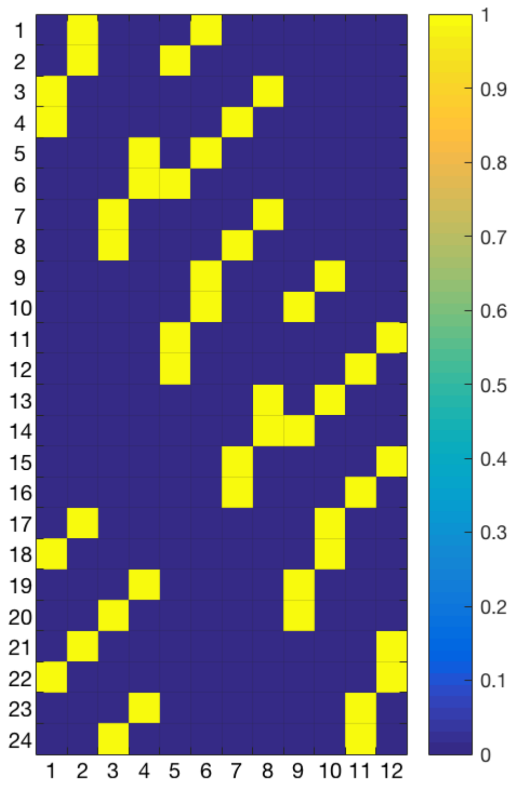


Fig. 5.18: Graphical representation of the matrix describing the connection between a cable and a "distant neighbor"

The first Tegotae term encapsulates the closer neighbors, i.e. those given in Figure 4.3 (right), whereas the second Tegotae term encapsulates the "distant" ones. Opposite signs are used for σ_1 and σ_2 as it is assumed that the contributions of both the closer and the further neighbors is also opposite in its nature.

5.3.2 SUPERball Matlab model

In order to implement the Tegotae rules, the ground reaction forces should be quantified. It was considered to do so in NTRT, but through discussion with other members of the community, it was discovered that attempts had already been made to quantify GRFs, but that no significant results had been obtained.

Subsequently, another simulation environment was required to test the algorithm. A preliminary Matlab model had been implemented in [23] to do state estimation of SUPERball. The basis for that model had been compared to the NTRT one and results were quite similar [24].

Using now this model would be beneficial, as normal forces computations had been incorporated into it. Subsequently, the model was adapted to compute CPGs online, whilst considering the Tegotae rules described in the previous subsection.

5.3.3 Parameters search

As there is no way of predicting accurate values that would allow SUPERball to achieve "inter-limb coordination", an approach using evolutionary algorithms would be used. Moreover, as the model being used for this implementation is within Matlab, the Global Optimization Toolbox may be used.

Two different evolutionary algorithms were selected, namely Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO). The first one has the benefit of also having a preliminary implementation within NTRT's Learning Library. Subsequently, when the issues described in subsection 4.6.1 will have been fully resolved, its results could then be compared to those obtained in Matlab. The second one has been selected as it would be useful to compare both algorithms to assess if one was effectively better than the other.

The process flow for the PSO algorithm was presented in Figure 5.1, and that of the GA algorithm is presented in Figure 5.19.

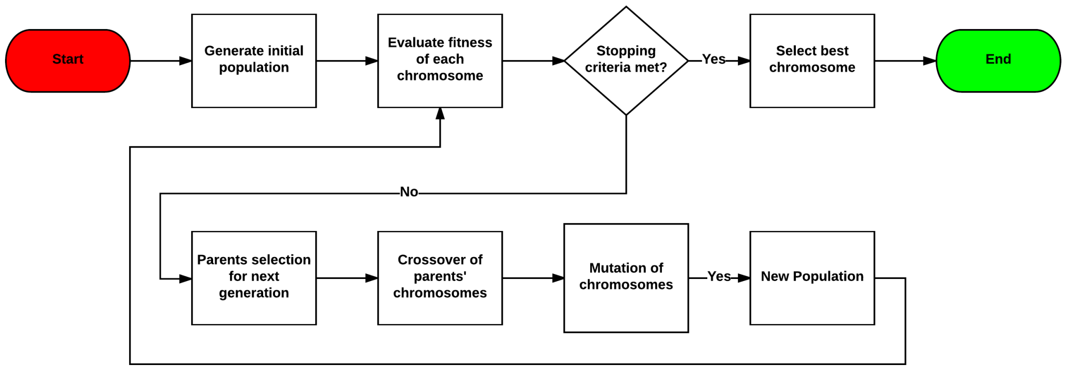

Start
Generate initial population
Evaluate fitness of each chromosome
Stopping criteria met?
Yes
Select best chromosome
End
No
Parents selection for next generation
Crossover of parents' chromosomes
Mutation of chromosomes
Yes
New Population

Fig. 5.19: Process flow for a Genetic Algorithm

To observe if this implementation would make a simple gait emerge, the phases obtained would be incorporated into the output as Equation 5.5. No antiphase mechanism was added to this subsection's model (unlike Equation 5.1), as it is assumed that it will inherently emerge from the GRFs modulation.

$$l_i = l_{i_0} \cdot (1 + \alpha_i \cdot \sin(\phi_i)) \tag{5.5}$$

The open parameters that would be tuned thanks to both algorithms are as follows:

- Common intrinsic frequency of the oscillators ω;
- Common coupling to the 1st Tegotae equation σ_1;
- Common coupling to the 2nd Tegotae equation σ_2;
- 24 amplitudes for the oscillatory rest length commands α_i.

The same cost function as in Equation 5.2 was used. When using a genetic algorithm, the results after a population of 150 evolved for 150 generations generated the values seen in Figure 5.20 and Figure 5.21.

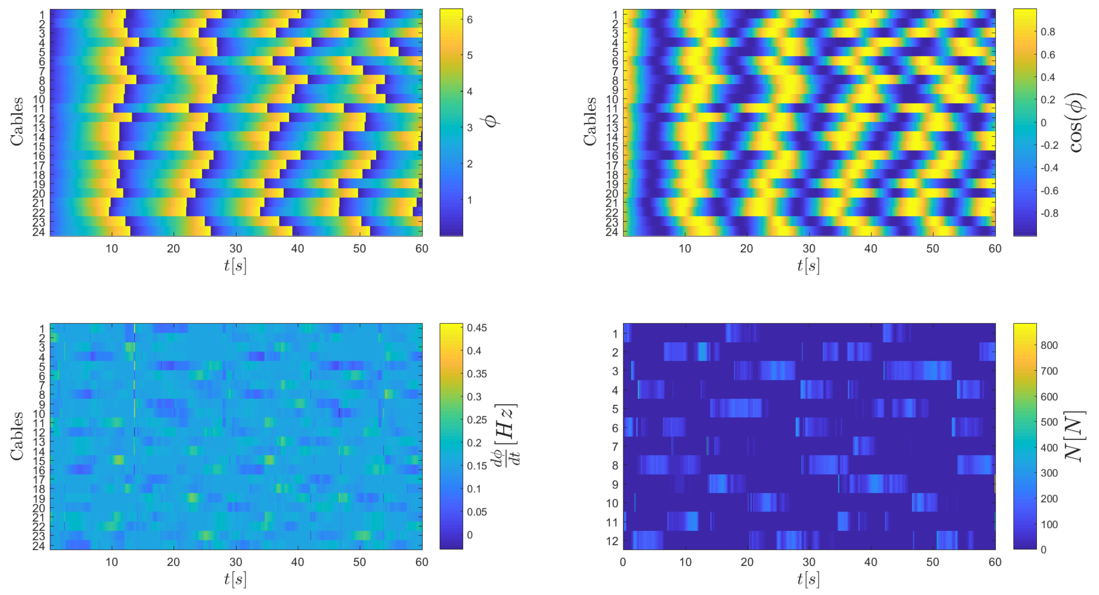


Fig. 5.20: Results obtained with the Matlab model on flat terrain after a genetic algorithm optimization

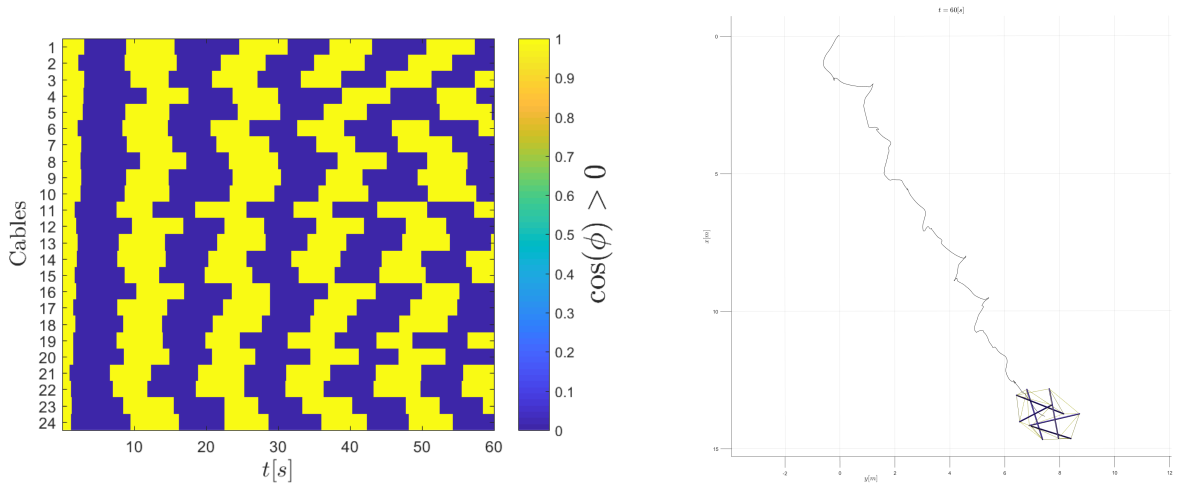


Fig. 5.21: Binary results (left) and aerial view of the path made by SUPERball (right) with the genetic algorithm results seen in Figure 5.20

It can particularly be seen in Figure 5.21 that after an initial stabilization cycle, a mostly periodic gait with a frequency of about $0.1[Hz]$ emerges. Certainly, it is not a perfectly periodic gait (as it can be seen for example for cable 4 around $t = 50[s]$), but there is clearly a trend. The video of this simulation can be found in https://youtu.be/7Z8rnM1mDxI.

Seeing that a general direction was being followed, it may be interesting to see how the system reacts when the intrinsic frequency of the oscillators is changed. Figure 5.22 to Figure 5.25 show firstly what happens when the frequency is reduced, then what happens when it is increased.

It can be seen that the path is the same as before, but that SUPERball hasn't traveled the same total distance. The intrinsic frequency thus seems to be directly controlling SUPERball's speed. The video can be seen at https://youtu.be/r8UWgKCZLnw.

Similarly, increasing the oscillators' intrinsic frequency allowed SUPERball to follow the same

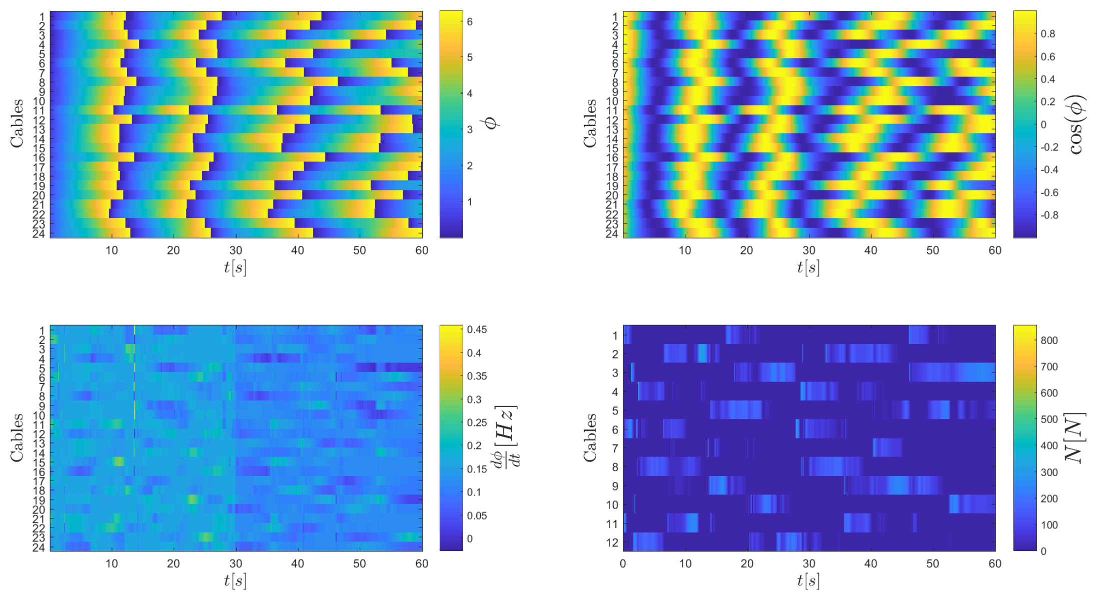


Fig. 5.22: Results obtained with the Matlab model on flat terrain after a genetic algorithm optimization. At $t = 30[s]$, the intrinsic frequency is changed to $^3/_4$ of the original one

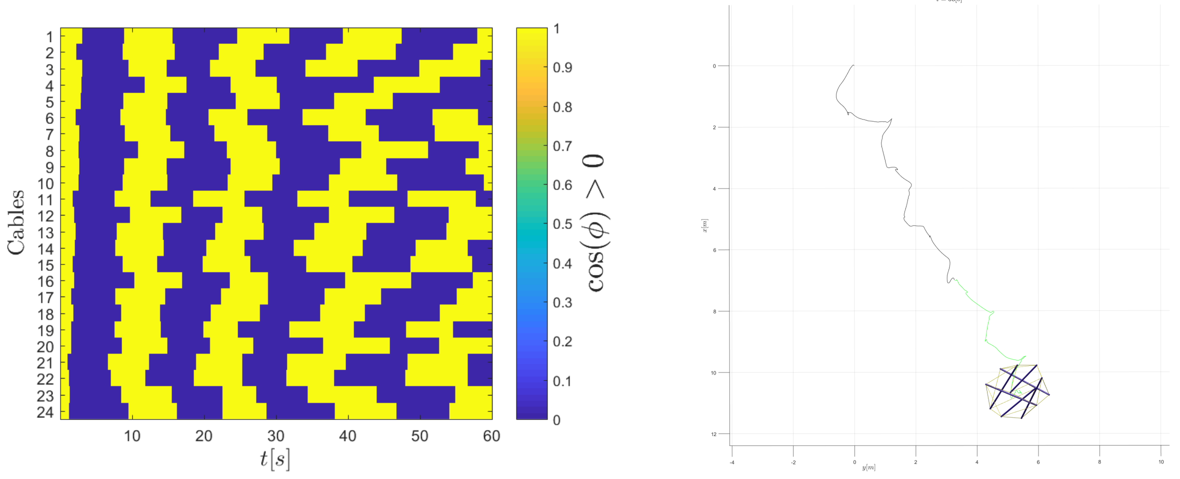


Fig. 5.23: Binary results (left) and aerial view of the path made by SUPERball (right) with the genetic algorithm results seen in Figure 5.22. At $t = 30[s]$, the intrinsic frequency is changed to $^3/_4$ of the original one (green path line)

path, and even go further than the original simulation. The accompanying video is located at https://youtu.be/tyZKy2Upljw.

In addition to running these optimizations with a GA, PSO was also tested. However, as Figure 5.26, Figure 5.27 and the video at https://youtu.be/QpIrnD9BqtQ indicate, the path obtained with it was not as efficient as the one obtained with the GA. Subsequently, the GA results should be the ones maintained for further research.

Other cost functions had also been tested, but the obtained results were not as cogent as the ones obtained with the minimalistic cost function. Indeed, it was thought that reducing the total variation the cables had to do throughout the simulation (for energetic cost minimization) or trying to force the signals to be phase-locked would contribute to the synchronization, but they were too restraining on the robot, even though they were set as multiple orders of magnitude below the overall distance traveled.

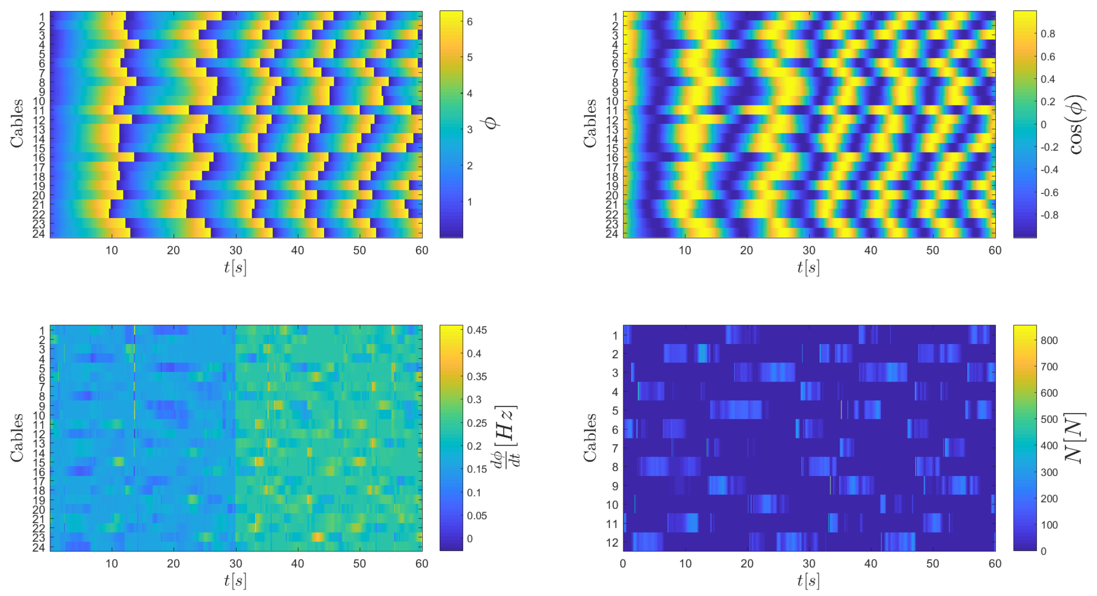


Fig. 5.24: Results obtained with the Matlab model on flat terrain after a genetic algorithm optimization. At $t = 30[s]$, the intrinsic frequency is changed to $^3/_2$ of the original one

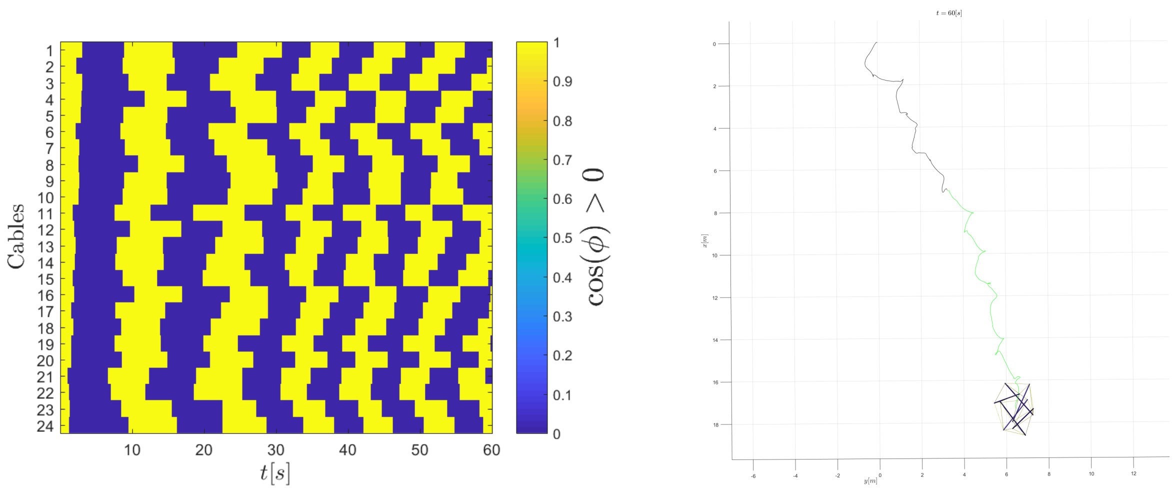


Fig. 5.25: Binary results (left) and aerial view of the path made by SUPERball (right) with the genetic algorithm results seen in Figure 5.24. At $t = 30[s]$, the intrinsic frequency is changed to $^3/_2$ of the original one (green path line)

The results obtained with the Tegotae rule looked very promising. Time constraints prevented further exploration of this implementation, for example testing it on uneven terrain. However, the code framework is present and for physical validation it has been discussed that one of the next steps for the prototype will be to integrate some type of force sensor into the end caps. This would allow to test the findings of this subsection not with commands computed offline, but with commands being generated online.

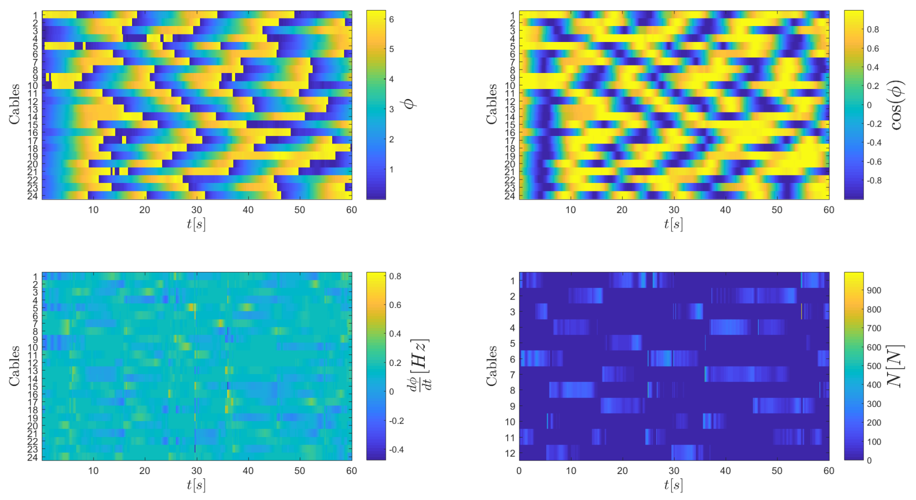


Fig. 5.26: Results obtained with the Matlab model on flat terrain after a particle swarm optimization

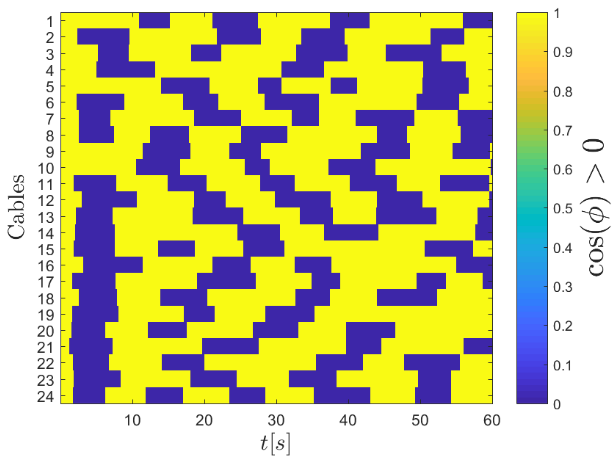
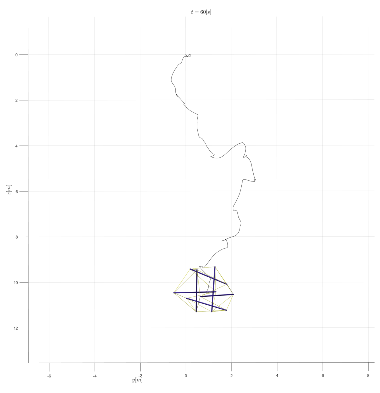



Fig. 5.27: Binary results (left) and aerial view of the path made by SUPERball (right) with the particle swarm optimization results seen in Figure 5.26

Chapter 6

Parallel Activities

6.1 Mockup models manufacturing

SUPERball v2.0 was being designed to have rods of approximately 1.9[m] in length. Being such a large robot, during the development phases, in order to have a better feeling of what the physical prototype would look like, it was decided to create mockup models of the robot. What's more, having smaller scale models is beneficial, as they can also be shown to visitors and NASA's management.

The mockup models were 30[%] scale models, which made them easily transportable. A few of them were manufactured in the spring of 2017, one of which can be seen in Figure 6.1.

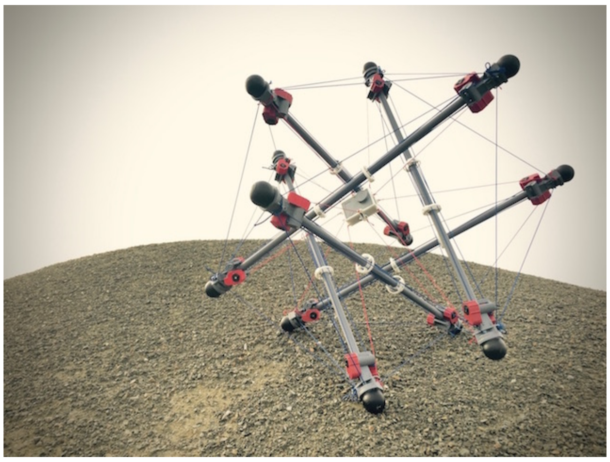


Fig. 6.1: SUPERball v2.0 mockup model made of 3D-printed parts, wood and nylon cables

These scaled models were manufactured simply using wood rods, 3D-printed PLA parts and nylon cables. All but the cables were painted such that the mockup more closely resembles the physical prototype, which had not yet been built as these were being manufactured.

Figure 6.1 is also interesting as it is promoting a more long-term view of the SUPERball project. There is indeed a payload within the the spherical tensegrity structure. In the case of the mockup, that payload is a GoPro© camera, as it is safe to assume that vision will be incorporated into future control algorithms for SUPERball or that a scientific instrument would have optics to some extent. What's more, having more cables should also have the added benefit of having a better distribution of unexpected forces across the whole robot.

However, the actuation method is not being showcased in the mockup model in the sense that the cables' spooling in the motors are not represented. But this design simplification can easily be justified during a presentation of the model.

6.2 SUPERball v2.0 prototype manufacturing

This subsection states in more detail SUPERball's main components, and points out a few non-trivial considerations one has to take when assembling the full robot.

As previously stated, SUPERball v2.0 is a tensegrity icosahedron. It is therefore consisted of 6 rigid rods and 24 tensile elements. The latter are actuated thanks to 24 Hebi Robotics©X8-3 motors[1]. These $460[g]$ BLDC motors are capable of providing a continuous torque of $3[N \cdot m]$, and have a maximum speed of $84[rpm]$. These values were used in chapter 4 as constraints on the simulations' actuators. What's more, they have an array of different sensors, such as an accelerometer, a gyroscope, temperature, voltage and current sensors embedded within each one of them. This makes them quite useful for trying out not only the controllers implemented by the author, but also other control algorithms, which may require different types of feedback. Figure 6.2 shows a Computer-Aided Design rendering of one of the robot's end caps.

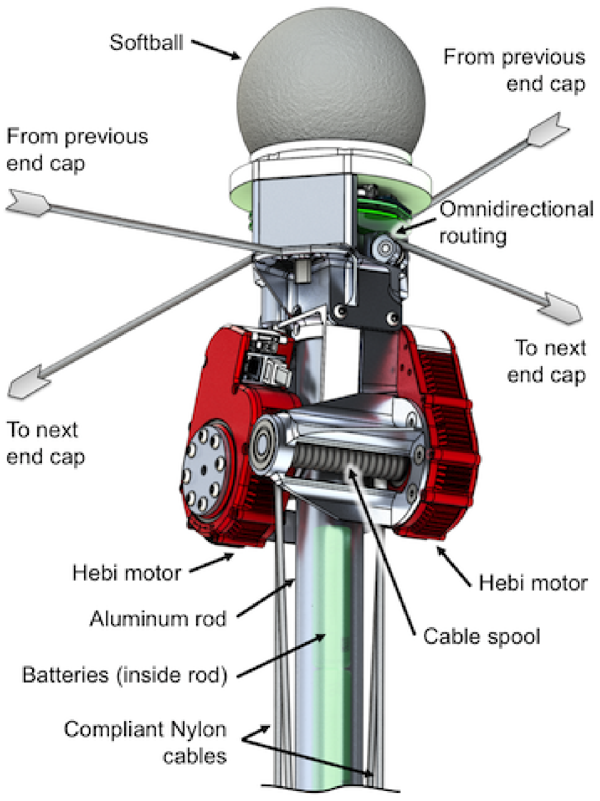


Fig. 6.2: SUPERball v2.0 end cap main components

[1]http://hebirobotics.com/products/

The cables are compliant nylon cables with a radius of $4[mm]$ and are capable of stretching up to $15[\%]$. The spools' housing have also been designed to be capable of reeling in up to $2[m]$ of cable, as not only locomotion control algorithms are of interest, but also those for folding/unfolding, which may require greater cable lengths for some of them, and lesser for others. Figure 6.3 illustrates that.

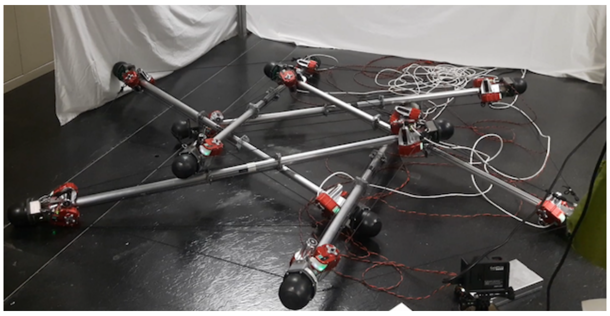


Fig. 6.3: SUPERball v2.0 folded into a star shape

In addition to it, the cables pass through an omnidirectional routing, which allows to reduce friction.

However, during assembly of such a tensegrity structure, one has to make sure that all cables have precisely the same length, otherwise the robot wouldn't be a symmetrical icosahedron due to the uneven distribution of tensions. The cables' routing has also been designed such that the spooling angle is always the same.

What's more, calibration is of extreme importance, especially since the motors only have a resolution of 8 turns. Seeing that reeling in or out the cables may change the number of cable layers around the motors' shafts, all of them are set such that the cables are exactly in between cable layers when it is in its stable configuration. Thanks to this, if cable is released, the it'll be at a lower layer, whereas if cable is reeled in then an additional layer will start. This was important as in the future, new onboard sensors shall be incorporated into the robot's prototype, particularly one verifying these cable layers, in order to complement the motors' encoders.

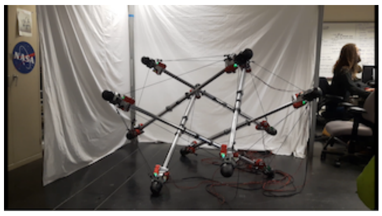
6.3 Testing a CPG with the manufactured prototype

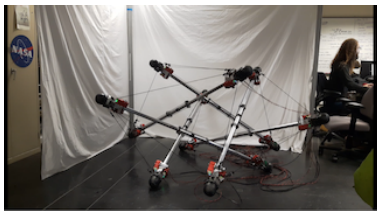
Right before the end of the author's stay at NASA ARC, SUPERball v2.0 had been assembled and initially calibrated.

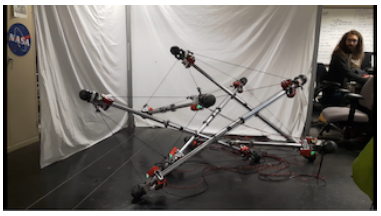
The control strategies detailed and implemented throughout this report could then be tested. However, due to time constraints, only one was possible. The selected one was the double chain of oscillators model, as the signal learner had not yet been provided with motor primitives data for a 24-actuator version of SUPERball, and because the robot did not have force sensors at its end caps to be used with the Tegotae implementation.

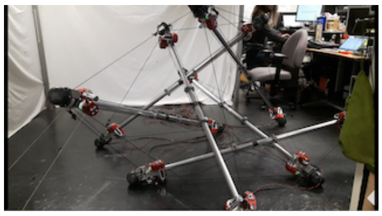
However, as is often the case when porting simulation results to actual physical prototypes, the simulation commands were not exactly reproduced by the robot. Some of the cables were retracting too much compared to what the simulation had computed. This certainly came from calibration errors and/or slight mismatches between the robot's components and their corresponding model's parameters.

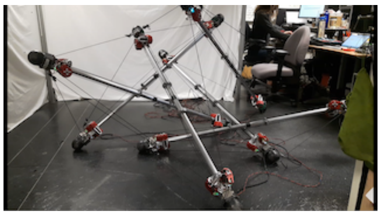
Still, some face transitions were obtained and can be seen in Figure 6.4. The full video can be seen at https://youtu.be/nhldoBLI96U.

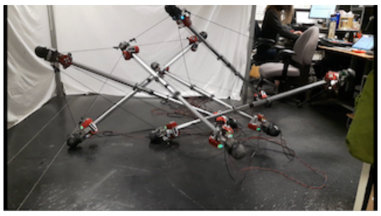

(a) At time $t = 0[s]$

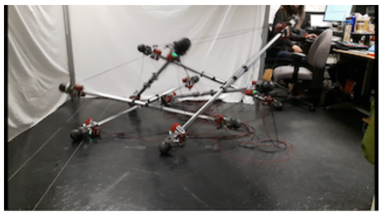

(b) At time $t = 7[s]$

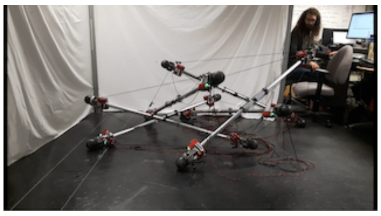

(c) At time $t = 11[s]$

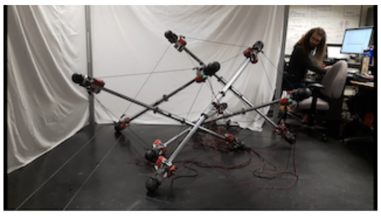

(d) At time $t = 17[s]$

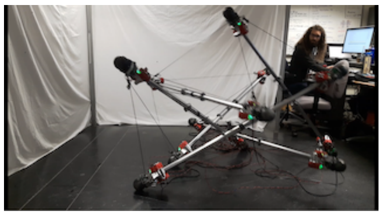

(e) At time $t = 45[s]$

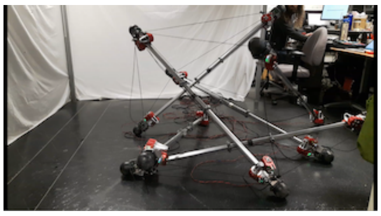

(f) At time $t = 48[s]$

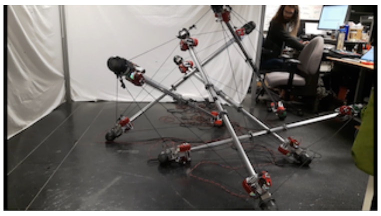

(g) At time $t = 50[s]$

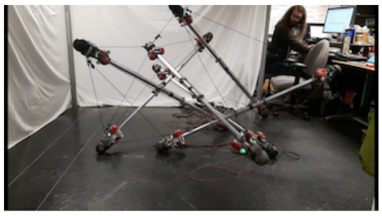

(h) At time $t = 53[s]$

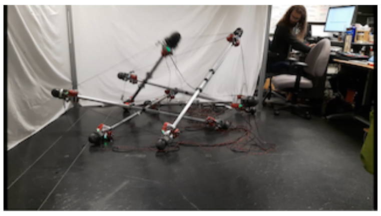

(i) At time $t = 76[s]$


(j) At time $t = 80[s]$


(k) At time $t = 83[s]$


(l) At time $t = 107[s]$


(m) At time $t = 117[s]$


(n) At time $t = 119[s]$

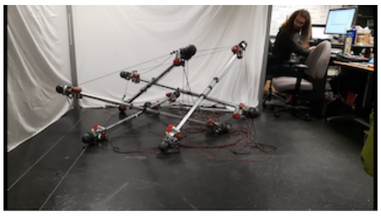

(o) At time $t = 123[s]$

Fig. 6.4: A few frames of the test of the double chain of oscillators CPG model, whose parameters have been optimized for flat terrain. For safety and testing purposes, the commands were being sent at a lower frequency than that of the simulation in Figure 5.3. The robot was correctly transitioning from one face to another, but calibration errors and the simulation/real world mismatch made that the tensions were not correct, therefore making the robot be much "lower" than what it is supposed to be

In the coming weeks, calibration of the robot will be improved and the obtained control system shall be tested once more.

Chapter 7

Conclusion

7.1 Future works

Results obtained throughout this project proved that different implementations of CPGs can be used to generate locomotion for a tensegrity icosahedron.

The double chain of oscillators model has generated an interesting "walking" gait, which was tested only once due to time constraints. Porting the results from the simulation environment to the physical prototype resulted in a few triangle transitions, but there were still some differences with the results obtained in NTRT. It would be of interest to continue working on this implementation, as it was shown that the gaits obtained were stable and periodic.

What's more, when the initial gait will be properly reproduced by the physical prototype, the frequency modulated versions will be able to be tested. Then, it will be interesting to test the results of the optimizations on uneven terrain. In fact, at NASA ARC it will possible to simulate the required rough terrain and even planetary surface missions thanks to the outdoors Roverscape test facility[1].

Moreover, steering for that model should be further explored. It wasn't able to have a full 360[°] control of its moving direction, but it could follow three different directions.

As it was stated in section 5.2, data for an offline machine-learning generated gait for a 24-actuator version of SUPERball had not yet been generated by the DTRL collaborators. Still, the signal-learning CPG framework implemented by the author should easily allow to create the CPG equivalents of the data. Then, it will be possible to test the obtained CPG thanks to the ROS interface.

Last but certainly not least, the Tegotae-based approach to CPGs should also be tested with the physical prototype. However, this requires creating and incorporating a GRF sensor to be added to each end cap of SUPERball. Alternatively, one could imagine computing the GRFs thanks to a neural network, which would be trained using the simulation results and computing the forces on the prototype with other sensors that are already present in the onboard electronics,

[1]https://www.nasa.gov/feature/roverscape

such as accelerometers and gyroscopes.

Much remains to be done, but with the talent of the people of the DTRL, there is no doubt that progress shall be achieved in the very near future.

All code related to the SUPERball project is located in NTRT's GitHub page. A new branch, containing all the code written by the author of this report between February and August of 2017, has been added[2].

7.2 Summary

Throughout the time that the author was carrying out his research at NASA ARC, the following contributions have been made by him:

- Created the new nomenclature for SUPERball v2.0 and updated the NTRT models;
- Implemented a double chain of oscillators model for SUPERball;
- Ran an optimization algorithm to tune the double chain of oscillators controller;
- Implemented a framework to learn arbitrary primitive motor signals;
- Created and implemented a Tegotae-based model for SUPERball;
- Ran different optimization algorithms to tune the Tegotae-based controller;
- Helped other members of DTRL in manufacturing and testing tasks.

7.3 Overall conclusion

Throughout this Master Project, it was studied if bio-inspired control techniques could be applied to a tensegrity robot to generate locomotion.

Though tensegrities in general can be used to model certain biomechanical elements, a complete tensegrity icosahedron does not exist in Nature (to the best of the author's knowledge). Therefore, there was no guarantee that achieving locomotion would be possible by using bio-inspired control techniques to command SUPERball.

However, all three different approaches were capable of making SUPERball move. CPGs have proven to provide effective feedforward (and partly feedback) components, even for such a highly complex and non-linear system that is that robot.

The work done in the scope of this project provides a framework to continue exploring these control strategies. Hopefully in the future many advantages will be found in using CPGs to control tensegrities. Being such a simple concept to grasp, yet capable of amazingly complex

[2]https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/tree/SUPERball_CPG

gaits, it can be assumed that future robotic applications will be using them more and more often.

It is very important to have robots with compliant structures, as recent robotics developments are highlighting the need of having robots that are easy and safe to interact with, as they become more and more present in our daily lives. Tensegrity structures being inherently compliant structures, it is safe to assume they will be used in numerous applications in the near future.

The author would like to thank once more all those that have made the opportunity to carry out this research project at NASA possible. It truly was a once in a lifetime experience and a dream come true for him. No better way could have been imagined to complete his engineering studies.

[illegible]

Marc André Leroy
NASA Ames Research Center
Moffett Field, CA, USA
Aug. 18th, 2017

References

[1] V. SunSpiral, A. Agogino, and D. Atkinson. *Super Ball Bot - Structures for Planetary Landing and Exploration.* Tech. rep. NASA Innovative Advanced Concepts (NIAC) Program, 2015.

[2] M. Masic and R. E. Skelton. “Open-Loop Control of Class-2 Tensegrity Towers.” In: *Proceedings of the International Society for Optics and Photonics (SPIE)* 5383 (2004), pp. 298–308.

[3] D. E. Ingber. “Cellular Tensegrity: Defining New Rules of Biological Design that Govern the Cytoskeleton.” In: *Journal of Cell Science* 104 (1993), pp. 613–627.

[4] T. Flemons. *The Bones of Tensegrity.* 2012. URL: http://www.intensiondesigns.com/bones_of_tensegrity.html.

[5] X. Geng, M. Zhang, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel, and S. Levine. “Deep Reinforcement Learning for Tensegrity Robot Locomotion.” In: *IEEE International Conference on Robotics and Automation (ICRA)* (2017).

[6] A. Cohen, P. Holmes, and R. Rand. “The Nature of the Coupling Between Segmental Oscillators of the Lamprey Spinal Generator for Locomotion: A Mathematical Model.” In: *Journal of Mathematical Biology* 13 (1982), pp. 345–369.

[7] R. Rand, A. Cohen, and P. Holmes. “Systems of Coupled Oscillators as Models of Central Pattern Generators.” In: *Neural Control of Rhythmic Movements in Vertebrates* (1988), pp. 333–367.

[8] S. Grillner and P. Wallén. “Central Pattern Generators for Locomotion, with Special Reference to Vertebrates.” In: *Annual Review of Neuroscience* 8 (1985), pp. 233–261.

[9] D. Bucher, G. Haspel, J. Golowasch, and F. Nadim. “Central Pattern Generators.” In: *Encyclopedia of Life Sciences* (2015), pp. 1–12.

[10] F. Delcomyn. “Neural basis of rhythmic behavior in animals.” In: *Science* 210.4469 (1980), pp. 492–498.

[11] A. J. Ijspeert. “Central pattern generators for locomotion control in animals and robots: A review.” In: *Neural Networks* 21 (2008), pp. 642–653.

[12] K. Karakasiliotis, R. Thandiackal, K. Melo, T. Horvat, N. K. Mahabadi, S. Tsitkov, J.-M. Cabelguen, and A. J. Ijspeert. “From cineradiography to biorobots: an approach for designing robots to emulate and study animal locomotion.” In: *Journal of The Royal Society Interface* 13 (2016).

[13] A. Sprowitz, A. Tuleu, M. Vespignani, M. Ajallooeian, E. Badri, and A. J. Ijspeert. “Towards Dynamic Trot Gait Locomotion - Design, Control, and Experiments with Cheetah-cub, a Compliant Quadruped Robot.” In: *International Journal of Robotics Research* 32.8 (2013), pp. 933 –951.

[14] F. Dzeladini, A. R. Wu, D. Renjewski, A. Arami, E. Burdet, E. van Asseldonk, H. van der Kooij, and A. J. Ijspeert. "Effects of a neuromuscular controller on a powered ankle exoskeleton during human walking." In: *IEEE International Conference on Biomedical Robotics and Biomechatronics* (2016), pp. 617–622.

[15] M. Ajallooeian, J. van den Kieboom, A. Mukovskiy, M. A. Giese, and A. J. Ijspeert. "A general family of morphed nonlinear phase oscillators with arbitrary limit cycle shape." In: *Elsevier Physica D* 263 (2013), pp. 41–56.

[16] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen. "From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model." In: *Science* 315.5817 (2007), pp. 1416–1420.

[17] D. Owaki, M. Goda, S. Miyazawa, and A. Ishiguro. "A Minimal Model Describing Hexapedal Interlimb Coordination: The Tegotae-Based Approach." In: *Frontiers in Neurorobotics* 11.29 (2017).

[18] K. Caluwaerts, J. Despraz, A. Iscen, A. P. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral. "Design and Control of Compliant Tensegrity Robots Through Simulation and Hardware Validation." In: *J. R. Soc. Interface 11* (2014).

[19] J. Bruce. "Design, Building, Testing, and Control of SUPERball: A Tensegrity Robot to Enable New Forms of Planetary Exploration." PhD thesis. University of California, Santa Cruz, 2016.

[20] F. Dzeladini, J. van den Kieboom, and A. J. Ijspeert. "The contribution of a central pattern generator in a reflex-based neuromuscular model." In: *Frontiers in Human Neuroscience* 8.371 (2014).

[21] M. LaBarbera. "Why the Wheels Won't Go." In: *The American Naturalist* 121.3 (1983), pp. 395–408.

[22] S. Levine and V. Koltun. "Guided Policy Search." In: *Proceedings of the 30th International Conference on Machine Learning* 28 (2013).

[23] K. Caluwaerts, J. Bruce, J. M. Friesen, and V. SunSpiral. "State Estimation for Tensegrity Robots." In: *International Conference on Robotics and Automation* (2016), pp. 1860–1865.

[24] J. M. Friesen, P. Glick, M. Fanton, P. Manovi, A. Xydes, T. Bewley, and V. SunSpiral. "The Second Generation Prototype of a Duct Climbing Tensegrity Robot, DuCTTv2." In: *IEEE International Conference on Robotics and Automation (ICRA)* (2016), pp. 2123–2128.

[25] J. van den Kieboom. *Optimization Framework Conceptual Overview.* 2014. URL: http://biorob2.epfl.ch/users/jvanden/docs/optimization-concept/concept.pdf.

[26] R. Motro. *Tensegrity: Structural Systems for the Future.* 2003.

[27] A. P. Sabelhaus, J. Bruce, K. Caluwaerts, P. Manovi, R. Fallah Firoozi, S. Dobi, A. M. Agogino, and V. SunSpiral. "System Design and Locomotion of SUPERball, an Untethered Tensegrity Robot." In: *IEEE International Conference on Robotics and Automation (ICRA)* (2015).

[28] A. Iscen, K. Caluwaerts, J. Bruce, A. Agogino, V. SunSpiral, and K. Tumer. "Learning Tensegrity Locomotion Using Open-Loop Control Signals and Coevolutionary Algorithms." In: *Artificial Life* 21 (2015), pp. 119–140.

[29] A. Iscen, A. Agogino, V. SunSpiral, and K. Tumer. "Flop and Roll: Learning Robust Goal-Directed Locomotion for a Tensegrity Robot." In: *International Conference on Intelligent Robots and Systems (IROS)* (2014).

[30] R. E. Skelton and M. de Oliveira. *Tensegrity Systems.* Springer, 2009.

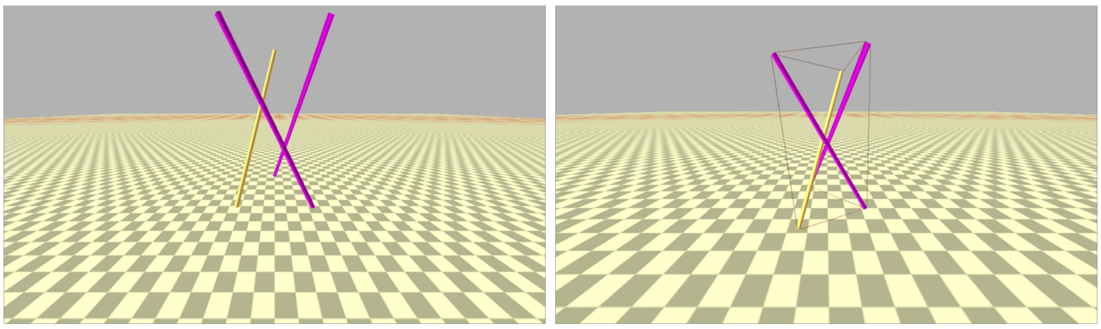
Appendix A

YAML/C++ Comparison within NTRT

A.1 Implementation in YAML

The following YAML code can be used to generate a simple 3-rod tensegrity structure within NTRT, which can be seen in Figure A.1.

```
nodes:
     bottom1:  [-5, 0, 0]
     bottom2:  [5, 0, 0]
     bottom3:  [0, 0, 8.66]

     top1:  [-5, 5, 0]
     top2:  [5, 5, 0]
     top3:  [0, 5, 8.66]

pair_groups:
     rod:
          - [bottom1, top2]
          - [bottom2, top3]
          - [bottom3, top1]

     string:
          - [bottom1, top2]
          - [bottom2, top3]
          - [bottom1, top3]

          - [top1, top2]
          - [top2, top3]
          - [top1, top3]

          - [bottom1, top1]
          - [bottom2, top2]
          - [bottom3, top3]

builders:
     rod:
          class:  tgRodInfo
          parameters:
               density:  0.688
               radius:  0.31
     string:
          class:  tgBasicActuatorInfo
          parameters:
               stiffness:  1000
               damping:  10
               pretension:  300
               max_tension:  70000.0
```



(a) At time $t = 0[s]$ (b) At time $t = 1[s]$

Fig. A.1: Output of the presented YAML code for a 3-rod tensegrity structure within NTRT at the initialization (left), and a few timesteps after (right)

A.2 Implementation in C++

The following C++ code would be the one require to create a 3-rod tensegrity structure. It can clearly be seen that YAML files should be used instead, thanks to their better readability.

```
#include "PrismModel.h"
#include "core/tgBasicActuator.h"
#include "core/tgRod.h"
#include "tgcreator/tgBuildSpec.h"
#include "tgcreator/tgBasicActuatorInfo.h"
#include "tgcreator/tgRodInfo.h"
#include "tgcreator/tgStructure.h"
#include "tgcreator/tgStructureInfo.h"
#include "LinearMath/btVector3.h"
#include <stdexcept>

namespace
{
     const struct Config
     {
          double density;
          double radius;
          double stiffness;
          double damping;
          double pretension;
          double triangle_length;
          double triangle_height;
          double prism_height;
     } c =
     {
          0.2, // density (mass / length3̂)
          0.31, // radius (length)
          1000.0, // stiffness (mass / sec2̂)
          10.0, // damping (mass / sec)
          500.0, // pretension (mass * length / sec2̂)
          10.0, // triangle_length (length)
          10.0, // triangle_height (length)
          20.0, // prism_height (length)
     } ;
} // namespace

PrismModel::PrismModel() :
tgModel()
{
}
PrismModel:: PrismModel()
{
}
void PrismModel::addNodes(tgStructure& s, double edge, double width, double height)
{
     // bottom right
     s.addNode(-edge / 2.0, 0, 0); // 1
     // bottom left
     s.addNode( edge / 2.0, 0, 0); // 2
     // bottom front
     s.addNode(0, 0, width); // 3
     // top right
     s.addNode(-edge / 2.0, height, 0); // 4
     // top left
     s.addNode( edge / 2.0, height, 0); // 5
     // top front
     s.addNode(0, height, width); // 6
}
void PrismModel::addRods(tgStructure& s)
{
     s.addPair( 0, 4, "rod");
     s.addPair( 1, 5, "rod");
     s.addPair( 2, 3, "rod");
}
void PrismModel::addMuscles(tgStructure& s)
{
     // Bottom Triangle
     s.addPair(0, 1, "muscle");
     s.addPair(1, 2, "muscle");
     s.addPair(2, 0, "muscle");

     // Top
     s.addPair(3, 4, "muscle");
     s.addPair(4, 5, "muscle");
     s.addPair(5, 3, "muscle");

     //Edges
     s.addPair(0, 3, "muscle");
     s.addPair(1, 4, "muscle");
     s.addPair(2, 5, "muscle");
}
void PrismModel::setup(tgWorld& world)
{
     const tgRod::Config rodConfig(c.radius, c.density);
     const tgSpringCableActuator::Config muscleConfig(c.stiffness, c.damping, c.pretension)

     // Create a structure that will hold the details of this model
     tgStructure s;

     // Add nodes to the structure
     addNodes(s, c.triangle_length, c.triangle_height, c.prism_height);

     // Add rods to the structure
     addRods(s);

     // Add muscles to the structure
     addMuscles(s);

     // Move the structure so it doesn’t start in the ground
     s.move(btVector3(0, 10, 0));

     // Create the build spec that uses tags to turn the structure into a real model
     tgBuildSpec spec;
     spec.addBuilder("rod", new tgRodInfo(rodConfig));
     spec.addBuilder("muscle", new tgBasicActuatorInfo(muscleConfig));

     // Create your structureInfo
     tgStructureInfo structureInfo(s, spec);

     // Use the structureInfo to build ourselves
     structureInfo.buildInto(*this, world);

     // We could now use tgCast::filter or similar to pull out the
     // models (e.g.  muscles) that we want to control.
     allActuators = tgCast::filter<tgModel, tgSpringCableActuator> (getDescendants());

     // Notify controllers that setup has finished.
     notifySetup();

     // Actually setup the children
     tgModel::setup(world);
}
void PrismModel::step(double dt)
{
     // Precondition
     if (dt <= 0.0)

          throw std::invalid_argument("dt is not positive");
```

```
    else

        // Notify observers (controllers) of the step so that they can take action
        notifyStep(dt);
        tgModel::step(dt); // Step any children

}
void PrismModel::onVisit(tgModelVisitor& r)
{
    // Example:  m_rod->getRigidBody()->dosomething()...
    tgModel::onVisit(r);
}
const std::vector<tgSpringCableActuator*>& PrismModel::getAllActuators() const
{
    return allActuators;
}
void PrismModel::teardown()
{
    notifyTeardown();
    tgModel::teardown();
}
```

Appendix B

NTRT App File example

The following C++ code shows the basic architecture of the source code required to create an NTRT app that shall be using a YAML file to create the tensegrity model.

```
#include "yamlbuilder/TensegrityModel.h"
#include "LengthControllerYAML.h"
#include "core/terrain/tgBoxGround.h"
#include "core/tgModel.h"
#include "core/tgSimulation.h"
#include "core/tgSimViewGraphics.h"
#include "core/tgWorld.h"
#include "LinearMath/btVector3.h"
#include <iostream>
#include <string>
#include <vector>

int main(int argc, char** argv)
{
     // For this YAML parser app, need to check that an argument path was
     // passed in.
     if (argv[1] == NULL)
     {
          throw std::invalid_argument("No arguments passed in to the application.
You need to specify which YAML file you would like to build.");
     }

     // create the ground and world.  Specify ground rotation in radians
     const double yaw = 0.0;
     const double pitch = 0.0;
     const double roll = 0.0;
     const tgBoxGround::Config groundConfig(btVector3(yaw, pitch, roll));
     // the world will delete this
     tgBoxGround* ground = new tgBoxGround(groundConfig);

     const tgWorld::Config config(98.1); // gravity, dm/sec2̂
     tgWorld world(config, ground);

     // create the view
     const double timestep_physics = 0.0001; // seconds
     //const double timestep_physics = 0.001;
     const double timestep_graphics = 1.f/60.f; // seconds
     tgSimViewGraphics view(world, timestep_physics, timestep_graphics);

     // create the simulation
     tgSimulation simulation(view);
```

```
    // create the models with their controllers and add the models to the simulation
    // This constructor for TensegrityModel takes the 'debugging' flag as the
    // second argument.
    TensegrityModel* const myModel = new TensegrityModel(argv[1],false);

    // Attach a controller to the model, if desired.
    // This is a controller that interacts with a generic TensegrityModel as
    // built by the TensegrityModel file.

    // Parameters for the LengthControllerYAML are specified in that .h file,
    // repeated here:
    double startTime = 5.0;
    double minLength = 0.7;
    double rate = 0.25;
    std::vector<std::string> tagsToControl;
    // See the threeBarModel.YAML file to see where "vertical_string" is used.
    tagsToControl.push_back("horizontal_string");

    // Create the controller
    Ctlr const myController = new Ctlr(startTime, minLength, rate, tagsToControl);

    // Attach the controller to the model
    simulation.attach(myController)
    // Add the model to the world
    simulation.addModel(myModel);

    simulation.run();

    // teardown is handled by delete
}
```

Appendix C

Optimized simulation parameters

This appendix states the results of the different optimization algorithms used in the CPGs' implementation.

ω_1	8.20368327e-01
α_1	4.94165304e-01
a_{13}	7.72563368e-02
a_{14}	-9.48981451e-01
ω_2	6.64453389e-01
α_2	4.99983857e-01
a_{31}	2.63744176e-01
a_{35}	-6.47762233e-02
a_{32}	4.40416848e-01
a_{36}	7.51836036e-01
ω_3	3.22210578e-01
α_3	-3.47760919e-01
a_{53}	-5.14416534e-01
a_{57}	-9.99998555e-01
a_{54}	7.01349754e-01
a_{58}	6.60893960e-01
ω_4	9.87293930e-01
α_4	-5.00000000e-01
a_{75}	-5.09037753e-02
a_{76}	-9.84507250e-02

Table C.1: Optimized parameters (using PSO) for the double chain of phase oscillators model on flat terrain

ω_1	0.13284348
α_1	0.15958025
a_{13}	-0.71523177
a_{14}	0.84594515
ω_2	0.4146127
α_2	-0.48740534
a_{31}	0.17690895
a_{35}	-0.38930058
a_{32}	0.87968614
a_{36}	-0.47189826
ω_3	0.78792464
α_3	-0.39499771
a_{53}	-0.28487358
a_{57}	0.54740467
a_{54}	-0.74158756
a_{58}	-0.26540554
ω_4	0.36496253
α_4	0.3580396
a_{75}	-0.55317575
a_{76}	-0.15056692

Table C.2: Optimized parameters (using PSO) for the double chain of phase oscillators model on hilly terrain

ω	0.989569545078806
σ_1	-0.002475745160053
σ_2	-0.001375942675489
α_1	0.125165812983512
α_2	-0.211605016093833
α_3	-0.211830301660209
α_4	-0.367278552730883
α_5	0.440640875218469
α_6	-0.386002749747558
α_7	-0.416039778620822
α_8	0.166581960684192
α_9	-0.307461797395952
α_{10}	0.244702155148751
α_{11}	0.408002088735525
α_{12}	-0.314583877276441
α_{13}	0.128888984146500
α_{14}	-0.259270273409962
α_{15}	0.461212444578856
α_{16}	0.261602920115001
α_{17}	-0.096403017295290
α_{18}	0.197595406930123
α_{19}	-0.337511736523481
α_{20}	-0.038249438263737
α_{21}	-0.287601134751449
α_{22}	-0.461235777576469
α_{23}	-0.476811373650674
α_{24}	0.376812706540407

Table C.3: Optimized parameters (using GA) for the Tegotae-based model on flat terrain

ω	0.977962865824894
σ_1	-0.001693474943499
σ_2	-0.007238718933995
α_1	-0.346258824547989
α_2	0.491568345493328
α_3	0.073170612902163
α_4	0.500000000000000
α_5	-0.317308555822357
α_6	-0.035712286768713
α_7	-0.231355392084874
α_8	-0.049867815237308
α_9	0.183031436454369
α_{10}	-0.115573105253081
α_{11}	0.427640410982165
α_{12}	-0.177189553155370
α_{13}	0.088259785990946
α_{14}	-0.423975558616757
α_{15}	0.000205723388772
α_{16}	0.455729335682354
α_{17}	0.216866167911571
α_{18}	-0.191516116367079
α_{19}	-0.305416134747032
α_{20}	-0.468784346236993
α_{21}	-0.253661294076273
α_{22}	-0.317156259831665
α_{23}	0.372571170190585
α_{24}	-0.241972725367681

Table C.4: Optimized parameters (using PSO) for the Tegotae-based model on flat terrain