

# Simulation of Biped Walking:

Implementing Taga's Model in PyODE

Juan Morago Guijarro

ASLAB-R-2008-005



The Autonomous Systems Laboratory  
*Universidad Politécnica de Madrid*



Master's Project Report  
**Simulation of Biped Walking:  
Implementing Taga's Model in PyODE**

Juan Morago Guijarro  
<[juanm313@gmail.com](mailto:juanm313@gmail.com)>

**Supervisor:** Associate Professor Örjan Ekeberg CBN  
**Examiner:** Professor Anders Lansner CBN  
Department of Computer Science and Communication  
NADA, KTH

## Abstract

As investigations about human walking are undertaken more and more often, new tools which can potentially be useful to authors must be evaluated. This project tries to measure how well PyODE physical library for Python performs under these circumstances. To do so, our first objective was to program a model created by Gentaro Taga using this library. We simulated this model, making the necessary corrections to generate stable gait. The system was subject to a series of experiments, so its performance could be further evaluated. Possible improvements were formulated, tried, and discussed. Results showed PyODE as easy to use and fast to program with. On the other hand, it led to a high tendency to numerical instability, which highly constrained the manipulation of the parameters of the system. This problem would often mean longer delays at investigations than the time it would save to use it, so we personally don't advice its use for this particular application.

## Acknowledgment

I would like to thank all the people who have contributed to this work in any way.

Some of these people, in Stockholm, are:

- Örjan Ekeberg, who was my supervisor and helped me in lots of ways guiding me trough all the long way.
- Martin Vopatek and Jan Olmårs, who were able to help me understand Linux and many other difficult stuff and made my time at NADA a very nice one.
- The rest of people working at NADA (KTH) for answering back lots of small questions and even for teaching me some biology.
- Cecilia Forssman, who was my coordinator with KTH, for helping me with a lot of little bureaucratic problems.
- All my roommates and other friends in Stockholm, for making me feel at home.

Some of these people, in Madrid, are:

- My family, for their support, in many different ways, during all this process, before it and after it.
- Teresa Riesgo Alcaide, Martine Day, Francisca Cepa Sado, Hugo del Valle Carrasco and all the staff at International Relations Department at ETSII (UPM), for making all this be possible.
- Ricardo Sanz, who was my supervisor in Madrid, for his advice.
- Alejandro Cordón Cubillo, Javier Lecanda (segundo apellido?) and Mario Simón Caballero, who helped me make this report look better.

# Index

1.- Introduction.....	1
1.1 Motivation.....	1
1.2 Objective.....	1
1.3 Human locomotion.....	2
1.4 Sensory system.....	3
1.5 Central pattern generators.....	4
1.6 Bipedal robots.....	6
1.7 Simulation models.....	7
1.7.1 Taga's model.....	8
1.7.2 Günther's model.....	8
1.7.3 Mochon & McMahon's model.....	9
1.8 Feet.....	9
1.9 Software tools.....	10
1.9.1 Python.....	10
1.9.2 ODE.....	10
1.9.3 PyODE.....	10
1.9.4 MATLAB.....	11
2.- Taga's model.....	12
2.1 Introduction to the model.....	12
2.2 The musculo-skeletal system.....	12
2.3 The global angle.....	14
2.4 The neural system.....	14
2.5 Generation of torques.....	15
2.6 The global states.....	15
3.- Development.....	17
3.1 Programming.....	17
3.2 Simulation environment.....	20
3.3 The neural model.....	21
3.4 The robot.....	22
3.4.1 Segments.....	22
3.4.2 Joints.....	23
3.5 The ground.....	24
3.6 Problems with first simulations (and solutions).....	24
3.6.1 The mechanical system had a high tendency to numerical instability.....	24
3.6.2 The system was unstable and, after a couple of steps, it fell to the ground.....	25
3.7 Corrections.....	26
3.7.1 Correcting force.....	26
3.7.2 Linkage of feet with the ground.....	26
4.- Simulations.....	28
4.1 Steady state.....	28
4.1.2 Simulation results.....	28
4.2 Simulating slopes.....	33
4.2.1 Progressive slope change.....	34
4.2.2 Sudden slope change.....	36
4.3 Gravity changes.....	37
4.4 Perturbations.....	38
4.4.1 Sudden short perturbation.....	38
4.4.2 Sudden increase of weight.....	39

4.5 Importance of the global angle.....	40
4.6 Changes in time step.....	42
4.7 Robustness of the model.....	44
4.7.1 Changes in the strength of neural connections.....	44
4.7.2 Changes in sensory inputs.....	44
4.7.3 Changes in impedance parameters.....	45
4.7.4 Changes in Rhythmic Force Controller's parameters.....	46
4.7.5 Changes in neurons' time constants.....	46
5.- Conclusion.....	47
6.- References.....	48
Annex A: How to create a stick figure.....	50

## Index of figures

1.1 Phases of gait.....	2
1.2 Flow of information.....	4
1.3 Half-centered model.....	5
1.4 Small perturbation.....	5
2.1 Interaction between sub-systems.....	12
2.2 Taga's mechanical model.....	13
2.3 Interaction foot-ground.....	13
2.4 Passive torques at the joints.....	14
2.5 Position of the feet in each global state.....	16
3.1 Neuro-musculo-skeletal system.....	18
3.2 Block grouping.....	18
3.3 Stick figure example.....	20
3.4 Simulation window.....	21
3.5 Classes, variables and functions at "Model".....	22
3.6 Comparison between "setGravity" and manual setting.....	25
4.1 Gait at steady walking.....	28
4.2 Steady gait: stick figure.....	28
4.3 Steady gait: global angle, global angle velocity and global states.....	29
4.4 Steady gait: torques by the neural network and the impedance controller.....	30
4.5 Steady gait: torques at the joints.....	31
4.6 Steady gait: angles of the segments.....	32
4.7 Steady gait: neural network's output.....	33
4.8 Progressive change of slope: evolution of the slope.....	34
4.9 Progressive change of slope: stick figures with slope increases.....	35
4.10 Progressive change of slope: stick figures with slope decreases.....	35
4.11 Sudden change of slope: evolution of the slope.....	36
4.12 Sudden change of slope: stick figures.....	36
4.13 Sudden change of slope: global angle.....	37
4.14 System's behavior with gravity increase.....	37
4.15 System's behavior with gravity decrease.....	38
4.16 Sudden force at HAT.....	39
4.17 Sudden increase of weight at HIP.....	40
4.18 Stick figure with global states non-dependent of global angle.....	41
4.19 Stick figure of steady walking without global angle.....	42
4.20 Effects of time step.....	43
4.21 Changes in sensory input.....	45





# 1.- Introduction

## 1.1 Motivation

Biped locomotion is currently considered as an interesting area in the scientific world for many reasons. Among them we can mention:

- Development of biped robots: control in this kind of robots can use anthropomorphic walking as a good source of information [8].
- Medical applications: improvement in detection techniques for walking difficulties[12], aiding the correction of these through the development of more efficient orthosis [24], and others.
- Cinematographic industry: more realistic animations of walking humans, developed using computer animation.
- Human evolution: computer simulation of walking can be used to trace the evolutionary changes in shape and movement that lead to the present human being from its ancestors [11].
- Sports science: a deeper knowledge on this area may serve to improve the techniques used by athletes.
- Neuroscience: investigations on biped walking are closely related to this field. Unexpected behaviors in the models may lead to discoveries about the interaction of the neural system with the environment.

Indeed, it is suggested that an interaction between the modeling of systems and physiological studies would be the “most fruitful way to proceed” [9].

## 1.2 Objective

Our main goal in this project is to implement and, if possible, to further develop the model proposed by G. Taga in 1995[2] using PyODE. The purpose of this is to check how well the physical library fits in this system.

One of the problems of Taga's model, due to the way it is formulated, is that it is not easy to modify, as it is constituted by a high number of closely coupled equations. Implementing it in ODE, which could compute an important part of the model itself, could diminish this problem at the expense of losing computing speed. This way the model could be made more transparent, easy to understand and possible to modify in future work in a faster way.

## 1.3 Human locomotion

A human being, as a biped walker, faces special difficulties when walking. Its gait is dynamic, meaning that the projection of the center of gravity (COG) is not always located within the base of support. This makes the system more complicated to control. “A walking system is generally supposed to be statically balanced if and only if its center of mass projects vertically inside the convex hull of its contact points” [On the stability of...].

Humans face an extra problem: its COG is generally located higher than the hip, increasing the instabilities it suffers due to its biped condition. It is, therefore, necessary to rely on a good control system keeping track of the gait.

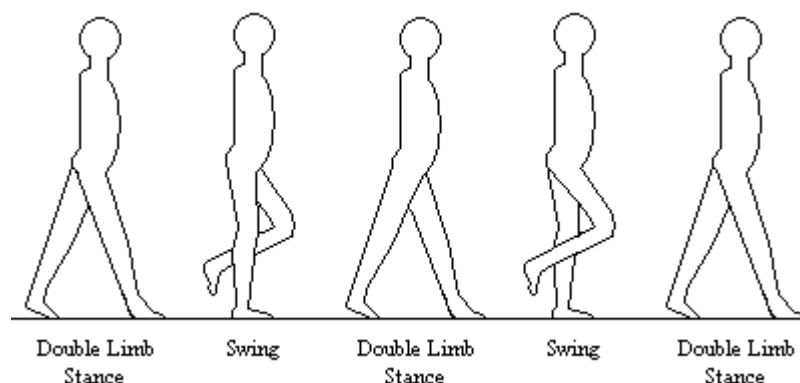
Gait is composed of several periodically repeated phases: initial double limb stance – single limb stance – terminal double limb stance – swing – double limb stance (see Fig. 1.1). Considering the movement as symmetric we can reduce the number of phases to two: double stance phase and single stance phase or swing.

The strategy followed during the swing phase is called “double pendulum” as it consists of:

- 1) a simple pendulum movement, made by the leg in the swing phase
- 2) an inverted pendulum movement, described by the rest of the body at the same moment

Both movements are constrained by the impacts of the swing leg with the ground and both pendulum's lengths are dependent on the current position. On the other hand, movement during the double stance phase may be simulated as a closed kinematic chain [20].

Frequencies of both pendular movements are related to physiological aspects such as the height. The smaller the person, the faster the frequency.



*Fig. 1.1: Diagram showing the different phases of human gait*

Energy consumption and risk avoidance are considered as the two main goals which lead to normal gait. It has been found that human beings choose a walking frequency such that the metabolic energy per unit distance is minimized [10]. In fact, locomotion has been described as “the translation of the center of gravity along a pathway requiring the least expenditure of energy” [27].

This is true once the central nervous system is mature and considering a flat ground with no obstacles. During the first few years of walking, though, minimizing the risk of falling is the main biomechanical strategy in children [27].

In order to achieve such a gait, some features are necessary [27]:

- 1) pelvic rotation
- 2) pelvic tilt
- 3) knee flexion
- 4) hip flexion
- 5) knee and ankle interaction
- 6) lateral pelvic displacement

The absence of one of these features may be corrected by the others, but the absence of two or more lead a to serious degeneration of walking [27].

## **1.4 Sensory system**

Animals use their sensory system to get information from the environment in order to move through it. This applies both to information about the route to follow and information about the immediate environment which is used to control the movement itself. Biomedical studies have proved that the stabilization of the head, where sight and inner ear are located, is one of the main objectives during locomotion [20].

Mammal's central pattern generators make use of three kinds of sensory receptors in order to correct gait. These receptors are: Golgi tendon organs, muscle spindles and cutaneous receptors [9].

Not all of these systems are necessary though, as there exists a redundancy among them, allowing walking to continue in case one or some receptors is inoperative[18].

Some gait models try to operate using these (or some of these) systems. Other models make use of sets of data that the human brain is not believed to have access to, such as the position of the joints or the angles of the limbs.

Sensory systems can be modeled by complex equations, but detection of discrete events is sometimes sufficient [9]. This is the case of Taga's model with ground contact of the feet and relative position of the legs.

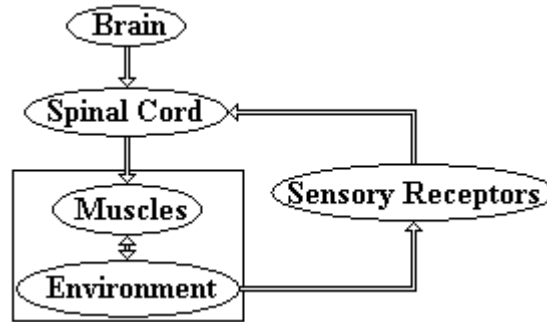


Fig. 1.2: Diagram representing flow of information between different components of the body during human locomotion

## 1.5 Central pattern generators

A central pattern generator (CPG) is a network of neurons, capable of creating a rhythmic output without a rhythmic input. They are present in all vertebrates, including humans. Scientific studies on patients with both complete and incomplete spinal cord injury have proven this [27]. CPGs exist since birth, but they are trained both by adaptation to the sensory input and by experience [19]. They are located in the spinal cord and constitute an essential part of the walking system, producing a pattern of signals which operate the muscles [15] and contribute to energy efficiency and risk avoidance [13].

As we mentioned before, CPGs use information from the sensory system. This is important as, even if they can produce walking patterns without this feedback, they are able to adapt the movement to the actual situation of the limbs [19]. This is essential for reaching stability, as has been proven in experiments with animals with cuts in the spine. These animals prove capable of walking on a treadmill, but not on their own [19].

CPGs are composed of “oscillators”:

“An oscillator is an autonomous dynamical system, i.e., a system of differential equations, with at least one limit cycle attractor. In other words, the solution of the system (after a transient time) is a closed cycle, which is asymptotically stable, i.e., if the system gets perturbed out of the limit cycle, then it returns back to it” [14].

Different models of oscillators exist. One of the simplest ones is the “Half-center model” (Fig. 1.3), with two self-inhibitory neurons. Each neuron of this model represents the effect of a group of neurons in nature [13].

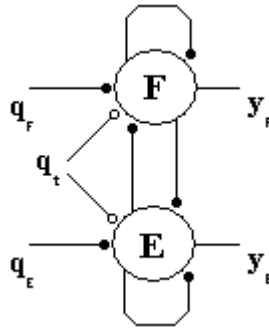


Fig. 1.3: Half-centered model of an oscillator

Neurons may be modeled with different degrees of complexity. The sigmoidal neuron model is formed by two non-differential equations. These neurons lack internal state and are difficult to train in recurrent networks [6]. Complexity increases gradually in other models. In the neural model used in Taga's model [2], each neuron is represented by two differential equations. Other neural models include three [1] or even more differential equations. As complexity increases and neurons become more realistic in biological terms, CPGs need smaller groups of neurons to achieve a given behavior [6], so an intermediate solution must be reached to minimize computing times.

As was mentioned before, oscillators are resistant to perturbations. If we apply a small perturbation, it will get the system out of the limit cycle and its distance from it will progressively get shorter until the system is in the limit cycle again. But the projection of the perturbation as a vector on the direction tangential to the limit cycle will translate into a phase difference [14]. This means the effect of a perturbation which is small enough not to turn the system unstable will result, after the system reaches the limit cycle again, in a constant difference in the time each point is reached.

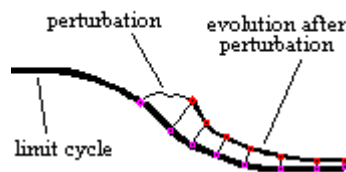


Fig. 1.4: Effect of a small perturbation on an oscillator

CPGs are an essential part of gait generation and many authors consider them as an interesting way to work within their models and simulations. Unfortunately, in many occasions hand tuning of the parameters of the CPG is necessary, due to extremely long computational times of automatic tuning methods [7].

## 1.6 Bipedal robots

Bipedal robots are mobile robots which fitting in human-oriented facilities and producing limited damages to the environment, due to their small area of contact with it [8]. Some uses could be medical, nursing and home-service robots [17]. They might also replace humans in hostile environments [24].

Most studies on humanoid robotics focus either on Zero Movement Point (ZMP) or on Passive Dynamic Walking (PDW). ZMP makes use of fixed trajectories. Calculation of such trajectories in dynamically stable walking is difficult, as dynamics are non-linear and highly complex [16]. This translates into many calculations in real time and hence requires powerful computing tools. Other problems of this method are a high consumption of energy and the fact that it doesn't ensure global stability [24].

The best way to achieve energy efficiency is to make the system work at its resonance frequency [13]. PDW robots generate gait by ballistic walking. They are able to reach stable walk on a fixed downslope path without energy consumption supplied by actuators. Gravity compensates for dissipative losses [7], which is mainly due to impacts between feet and ground [20], so actuators are necessary just for starting and breaking [25].

PDW robots are very sensitive to perturbations, just as to the starting position [23]. In spite of that, they can change their speed and lengths by adjusting their control inputs [5].

A deeper knowledge on bipedal robots may lead to a deeper knowledge on human being's walking system. Biped robots find human beings as their main influence, as it is "the most sophisticated and versatile biped known to man" [20]. Both disciplines find computer simulation of gait as a common tool, even though it has been criticized by some authors arguing biped robots are too complex to be completely simulated [4].

Some interesting biped robots are:

- ASIMO (Honda):  
<http://world.honda.com/ASIMO/>
- WL-10RD (Waseda University):  
[http://www.humanoid.waseda.ac.jp/booklet/kato\\_4.html](http://www.humanoid.waseda.ac.jp/booklet/kato_4.html)
- M2 (MIT Leg Laboratory):  
<http://www.ai.mit.edu/projects/leglab/robots/m2/m2.html>
- BIP2000 [26] (BIP-INRIA Grenoble, LAG-CNRS Grenoble, LMS-CNRS Poitiers, LMP-CNRS Poitiers):  
<http://www.inrialpes.fr/bip/>

These are just some examples. A much more complete list of bipedal robot projects may be found in the following web-site:

<http://www.mel.go.jp/soshiki/robot/undo/kajita/bipedsite-e.html>

## 1.7 Simulation models

Energy consumption of models simulating human walking tends to be much higher than that consumed by human beings. Low efficiency, especially when using some control methods, may increase it to twice those values[5] or even larger ones.

As a system turns more realistic in biological terms, computation requirements increase. Programming time increases too. For instance, tuning parameters in realistic neural models becomes more difficult [6].

On one hand, biologically accurate models lead to more efficient gaits. On the other hand, the degree of accuracy is limited by the available technology. Also, some authors may try to find completely different models, able to perform better than biological ones. But these last kind of models are out of the scope of this work, so we will not consider them.

This way, we can say that each different investigation must reach a compromise between biological accuracy and feasibility. Linked to this, we must say that development of technology has indeed influenced the models, making it possible to achieve more successful biologically accurate models in the last years [25] [28], comparing them with older ones [2] [10].

H. van der Kooij et. al. propose the following classification of “different approaches used to synthesize bipedal gait” [5]:

- 1) Open-loop control: input is calculated in order to achieve a set of prearranged trajectories. Solution fits only for a given environment in absence of perturbations and high computation is required.
- 2) Trajectory control: the trajectory of each joint is previously described and movement is corrected in order to keep it.
- 3) Set-point control: similar to the former method, but only a set of values at specific times are supplied.
- 4) Nonlinear dynamic system/ballistic walking approach: gait emerges from the interaction of the environment and the dynamical properties of the system. These systems are stable for small perturbations, but not necessarily for bigger ones.

Ballistic walking is limited to low speeds. If speed is increased, it reaches a point when the swing leg crashes the ground. It is necessary then to resort to muscle torques to avoid this and continue accelerating [10].

### - 3D-walking Vs 2D-walking

Some authors run their simulations on 3D environments [9] [11] [15] while others use 2D ones [22] [23] [2] [10].

It seems logical to think that 3D systems should be, in general, more accurate when trying to simulate gait. Unfortunately, the human locomotory system is large and, as far as we know, very complex [27], so calculations require a high amount of computation. 2D systems use to imply much less computation to solve each step, defined as the time passed for each new computation of the position of the body, the new muscle forces, etc.

Choosing the number of dimensions of our system is a very important decision to take and some factors should be taken in consideration, such as how much we gain in our simulation accuracy in expense of extra time, due to development and running. This is not an easy question and experience of the researcher will be the main tool to answer it.

### **1.7.1 Taga`s model**

Gentaro Taga is one of the people having created a model to study biped walking. We can find a good explanation of this model in Taga's paper [2]. It consists in a musculo-skeletal model (a simulation of a mechanical body and an environment) closely interacting with a neural model, consisting of neural oscillators forming a Central Pattern Generator (CPG). Gait naturally emerges when these two systems are connected.

The CPG gets information from the environment and, so, adapts to the circumstances, which increase the stability zone of the system and is probably the case in mammals[12].

A musculo-skeletal model is a set of links connected by joints and simulating the body dynamics and the muscular action over both body and environment.

Taga's model is analyzed more deeply in chapter 2.

### **1.7.2 Günther's model**

Michael Günther created a two-dimensional eleven-segment musculo-skeletal model for bipedal walking published in year 2002 [25].

Coordination in this model is not achieved by a CPG. Instead, a control algorithm is in charge of it. There is a set of two alternating neural states. At a given moment each leg is in a different state and they switch when the model evolves to a given position (concretely when the rear foot passes the other).

The musculo-skeletal model used by Günther is closer to human physiology than Taga's. Fourteen musculo-tendon pairs per leg are modeled as massless threads. Feet are composed of three segments and present a toe and two modeled ligaments.

The main advantage of this model is that it is more similar in biological terms to



human body. This quality is considered by many authors as a very desirable one when trying to reproduce human walking.

### **1.7.3 Mochon & McMahon's model**

The ballistic model by S. Mochon and T. McMahon [10] studies just the swing phase of walking. It is assumed that muscles act only during the double stance phase, so no muscular torque is applied during the swing phase (which implies no torque is applied in the model).

The model is two-dimensional and has three links. One of the links corresponds to the stance leg, and the other two to the swing leg. The foot of the stance leg is represented, but is only used to fix the position of its corresponding ankle by allowing just one rotational degree of freedom. The foot belonging to the swing leg can be considered as welded to the shank of the same leg (this is the reason why it is not considered as a separated link).

Ballistic walking results from the interaction of the system with the environment [5].

## **1.8 Feet**

Feet deserve a special mention in this brief introduction to biped walking. They are still one of the biggest difficulties most studies on this area must face. A quick look at a set of robots and simulations is enough to realize how each author solves this in his own different way. These include single-segmental feet [10] [16], two-segmental feet [2], three-segmental feet [25], circles [23] [7], wheels [24], human-like shaped feet (welded to the shank [4] or not [22]) and others.

A correct design of the feet is more important in biped robots than in those with multiple legs, as more legs means less weight in each of them. On the other hand, single-legged robots don't require any complex foot because of their bouncing kind of movement.

In biped robots, foot-ground impact is still one of the main problems during the control design, limiting maximum horizontal velocities [20]. A relatively high amount of energy must be absorbed by the heel so the foot does not bounce when it hits the ground.

The foot must be able to gradually transmit the pressure on the ground (COP) from the heel to the toes, during the swing phase, in order to produce a movement similar to the one of human beings. Toes are in fact a critical point in the design of feet, as their absence impedes to reach smooth, energy efficient movements, just as it happens with people who have suffered amputations of one or some of their toes.

Concluding, authors keep on looking for simple models of feet, capable of fulfilling the necessities of their models. The more an investigation looks for a biologically accurate kind of movement, the closer their design must be to human feet. As time passes, studies will probably try to reach even more realistic movements and, as a result, better models of human feet will have to be produced. And, maybe, new discoveries about our feet will be concluded from these studies.

## **1.9 Software tools**

Different tools were used to produce simulations of the system and to obtain and analyze data from these simulations. The most relevant ones are briefly introduced here.

### **1.9.1 Python**

Python is a programming language which offers us three advantages for working with it on this project. The first one is that it is object oriented. The second advantage is that it does not need to be compiled every time you make a change, making the process of programming faster. There is a third reason for using Python: it is based on C, which is a programming language already known by everybody involved in the project (so is Python in many cases).

### **1.9.2 ODE**

Open Dynamics Engine is a physics engine. It is a library containing functions which allow us to simulate physics models, constructing and solving the necessary differential equations. The reason why we are going to use ODE in this project is because it is stable, known by most of the people involved in the project and it is open-source. Another important reason is that it has been the program used in related projects in the department, so it has already proved to fit in similar circumstances.

More information about The Open Dynamics Engine (ODE) can be found at ODE's official web-page (<http://www.ode.org/>).

### **1.9.3 PyODE**

PyODE is a module that enables to use ODE in programs made for Python. We find this definition at PyODE's official web-page<sup>1</sup>:

---

<sup>1</sup> <http://pyode.sourceforge.net/> 15-VI-2007

“PyODE is a set of open-source Python bindings for The Open Dynamics Engine, an open-source physics engine.”

The reason why we use PyODE in this project is because it makes it much easier to work both with Python and ODE.

#### **1.9.4 MATLAB**

MATLAB is both a programming language and a computing environment. It is based on ANSI-C and includes a large amount of procedures (called functions) oriented to calculus, plotting, statistics, system control, and many others.

This application was used to print the diagrams of this project for several reasons:

- It was already a known language for everybody involved in the project.
- Programming in MATLAB is a relatively fast and easy task at an expense in computation time, which was not a problem as it was still short.
- Plotted diagrams using MATLAB are neat and easy to scale.
- MATLAB's computing environment was able to correctly interpret data from our program using a minimal amount of code.

## 2.- Taga's model

### 2.1 Introduction to the model

Gentaro Taga is one of the people who have created a model to study biped walking. We can find a good explanation of this model in Taga's paper [2]. In this project, we will implement this model in PyODE and, afterwards, we will subject it to experiments. A brief introduction of the model is presented in this chapter.

The model consists of a musculo-skeletal system (a simulation of a mechanical body and an environment) closely interacting with a neural system, which consists on a set of neural oscillators forming a Central Pattern Generator (CPG). Gait naturally emerges when these two systems are connected as shown on Fig. 2.1.

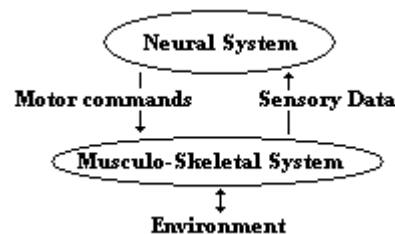


Fig. 2.1: Diagram of the interaction between the different sub-systems in Taga's model

### 2.2 The musculo-skeletal system

The musculo-skeletal model chosen by Taga is represented in Fig. 2.2. It is composed of eight segments or, more precisely, six segments plus the feet.

The HAT represents the head, the arms and the trunk, all together, in a single segment. This is the only segment of the body (not considering the feet as segments) with its COM displaced from its geometric center.

#### **-Feet:**

Feet are represented by triangles of appropriate proportions, but interaction between the musculo-skeletal system and the ground takes place only at the heels and the toes. At each foot, the inferior segment has the only purpose of closing the foot as a rigid solid.

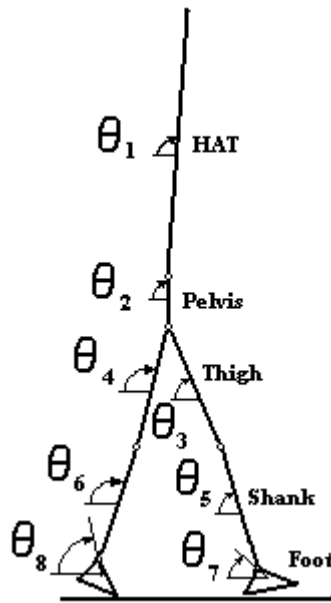


Fig. 2.2: G. Taga's mechanical model for human locomotion

**-The ground:**

The ground is modeled as a combination of springs and dampers at the heels and the toes, equivalent to the system represented at Fig. 2.3.

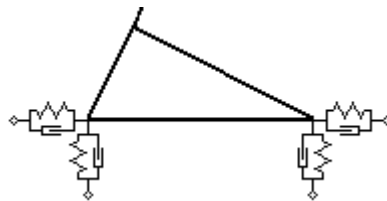


Fig. 2.3: Interaction foot-ground simulated by springs and dampers

**-Actuators**

The model relies on twenty muscles, even though they are not physically designed. Instead, they are conceptually represented by a series of equations. The forces in the muscles are combined to produce a single torque applied in each joint.

This way, even though the system has conceptual muscles, it should be better considered as a robot with a single motor in each joint. Actuators are controlled by the neural system and muscle forces are functions of its output (u).

**-Limits in flexion and extension:**

A series of torques are exerted to keep the angles of the joints within some limits. These torques represent both springs and dampers, as shown in Fig 2.4.

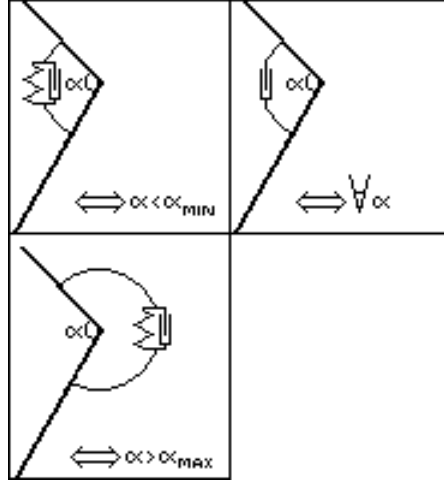


Fig.2.4: Passive torques exerted at the joints in Taga's model

## 2.3 The global angle

Global angle is defined as the angle existing at a given moment between the horizontal plane and the line containing the COG of the body and the COP (center of pressure) of the feet in contact with the ground.

There is a parallelism between this angle and that of an inverted pendulum and Taga uses it, in a similar way, as a feedback for the control system of his model. This is done in two ways. First, global angle is used as part of the definition of the states of the system. In other words, transitions between some states occur when global angle passes over a given value. Also, the sensory input of the neural system is dependent of the global angle and the global angle velocity.

## 2.4 The neural system

The “Neural Rhythm Generator” is composed of seven neural oscillators with two neurons for each of them. There are two neurons for each joint, controlling the muscles that operate it in each direction.

$$\begin{aligned}
 \tau_i \dot{u}_i &= -u_i - \beta f(v_i) + \sum_{j=1}^{14} w_{ij}^0 f(u_j) + u_0 + Q_i + S_i, \\
 \hat{\tau}_i \dot{v}_i &= -v_i + f(u_i), \\
 (f(u) &= \max(0, u)), (i=1,14)
 \end{aligned} \tag{2.1}$$

$u_i$  and  $v_i$  define the current state of each neuron of the  $i$ th oscillator.  $\tau_i$  and  $\hat{\tau}_i$  are

time constants related with the adaptation speed of the  $i$ th neuron.  $\beta$  is a constant also related with the adaptation effect.

$Q_i$  is included in the formula to represent the variation in the inter-neuronal weights in each state.

$$Q_i = \sum_{k=1}^6 S_{gk} \sum_{j=1}^{14} w_{ij}^k f(u_j) \quad (2.2)$$

$S_i$  represents the direct influence of the sensory system on the neural system (apart from that exerted in the neural coupling by the state). It includes the effect of stretch reflexes.

In summary, the neural system has the following inputs:

- $u_0$  : it has a constant value in all the experiments, even though its value is not the same in all of them. An increase in  $u_0$  translates in an increase in walking speed.
- $S_g$  : for a constant  $S_g$  (while the robot does not evolve to the next state), the state of the set of neurons will follow a progression depending only on itself (as we consider  $u_0$  as constant). This progression's objective is to lead the system to the next state in an appropriate way.
- The global angle: it affects  $S_i$  with the objective of contributing to the coordination between the neural and musculo-skeletal systems. It affects  $S_g$  too.
- $\theta_i$  : the angle of each segment of the robot is required as an input to  $S_g$ . This set of data is not accessible to neural system during human gait, but it is only used to simulate stretch reflexes, which are present in human being.

## 2.5 Generation of torques

The state of the oscillators ( $u$ ) is translated to adequate torques by the ‘‘Rhythmic Force Controller’’. The result is represented by  $T_{mri}$  and is a polynomial function for each different state. These torques are added to  $T_{mii}$ , which represents the stiffness and viscosity of the joints. The result of this addition is the torque exerted by each of the twenty muscles.

$$T_{mi} = T_{mri}(u, s_g) + T_{mii}(\theta, \dot{\theta}, s_g), (i=1,20) \quad (2.3)$$

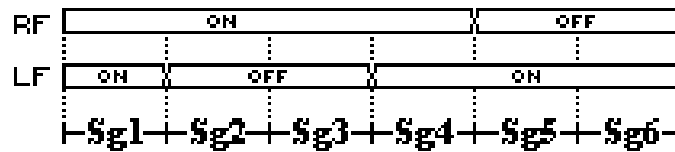
## 2.6 The global states

The global states divide the gait into a cyclic sequence of discrete phases. In each of

these phases, the system follows a different strategy. They are defined with two main purposes:

- 1) to maintain a dynamic linking between the musculo-skeletal and neural systems
- 2) to alter neural coupling in order to achieve complex behavior with an appropriate timing

There are six possible states of the system. These states are dependent on the position of the feet relative to the ground (see Fig.2.5). The state depends on the global angle too. The global angle marks the transitions from Sg2 to Sg3 and from Sg5 to Sg6 when it reaches a given value ( $90^\circ$  in both cases).



*Fig. 2.5: Contact of the feet and the ground during the global states. RF is for the right foot and LF is for the left foot*



## 3.- Development

### 3.1 Programming

The root of the programming process was a small program prepared by Professor Örjan Ekeberg (working at NADA, KTH, Stockholm and tutor of the project) with the purpose of helping students to get in contact with the programming environment. This program was used with his permission. As a starting point, it was extremely helpful and allowed to produce graphical simulations in a short time.

Ö. Ekeberg's program was composed of eight files containing the basics for a simulation using PyODE. All these files have, to a greater or lesser extent, being altered during the process of programming of this project, with the exception of `transform.py`, which remains untouched. Also, the file controlling this graphical interface remains almost unchanged.

The process of programming was divided in several sub-phases:

- 1) choosing the tools
- 2) deciding the structure of the program
- 3) writing the necessary code to produce simulations
- 4) deciding which set of data was interesting to get from the different simulations and writing new code to get this data
- 5) deciding which experiments should be done
- 6) writing the necessary code to plot the figures

These sub-phases will be briefly explained in a deeper detail now.

#### **Phase 1: Choosing the tools**

Some of the tools used for the code were formerly presented in the introduction. These include Python programming language, PyODE library and MATLAB. PyGame and OpenGL were used for the simulation as well.

#### **Phase 2.- Deciding the structure of the program**

Several structures were discussed, but quickly one of them prevailed. The selected structure is inspired by the block system proposed by G. Taga in his work [2] represented in Fig.3.1.

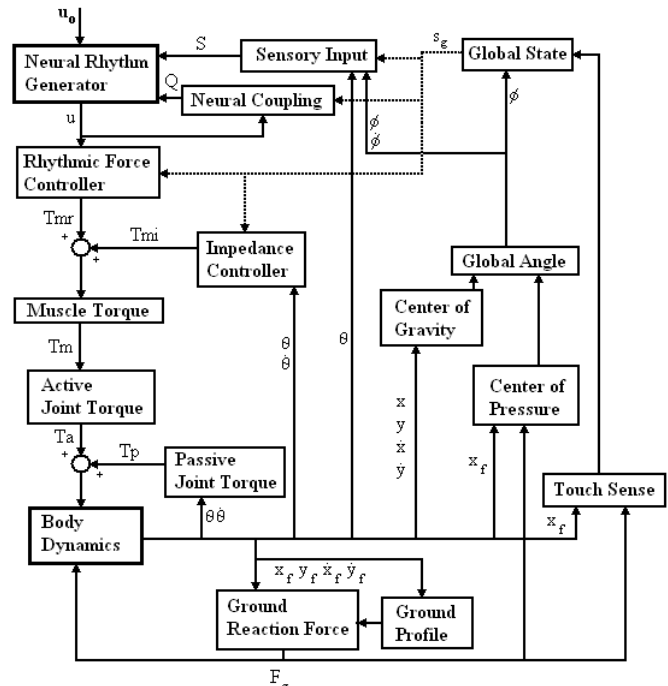


Fig. 3.1: Block diagram of the neuro-musculo-skeletal system for human locomotion in Taga's model

This was programmed using the procedure "step\_man" (file step\_man.py) which runs every block of the system once, in an appropriate order, each time it is called. This procedure is cyclically executed from the main file of the program, main.py.

Each block was simulated by a single file (with some exceptions listed below). Files were grouped in four folders (G1 to G4) to ease their location (Fig. 3.2). Execution flow in step\_man runs all the functions in each group before continuing to the next one, with the objective of making first reads easier.

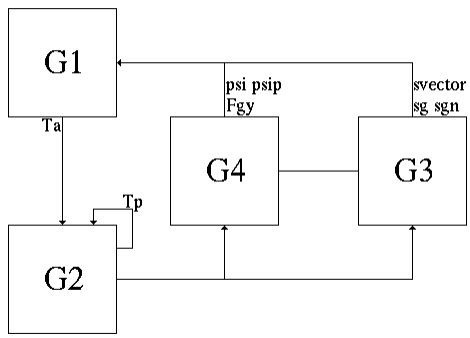


Fig. 3.2: Block diagram of the neuro-musculo-skeletal system for human locomotion in our model. Each of the blocks is composed by the procedures of the files included in the folder of the same name

### **Phase 3.- Writing the necessary code to produce simulations**

In order to reproduce the model by G. Taga, most of the model's equations were included in the code. The only exceptions are the equations representing the dynamics of the body and the environment, which were substituted by ODE engine. All this was integrated with the simulation engine, introducing necessary changes here.

### **Phase 4.- Deciding which set of data was interesting and writing new code to get this data**

A look on different papers of related work reveals a lack of homogeneity in the experiments simulations are subjected to. This may partially be explained by the different disciplines of the authors and, consequently, their different immediate interests. For instance, if we compare the work of an engineer dealing with robotics with the work of a biologist we can find they choose very different sets of data.

Our decision tried to fulfill two goals:

- 1) to give enough information about the simulation, so its degree of biological accuracy could be easily appreciated by an expert eye
- 2) not to give too much redundant information

An interesting consideration here was that the data should later be compared with that from Taga's model in the different experiments present on the studied papers [2] [3].

### **Phase 5.- Deciding which experiments should be done**

Once the robot proved able to walk with a stable gait on a flat surface with absence of perturbations, the next natural step was to check how well it performed on different situations. Gait initiation and ending was out of the scope from the beginning, so these kind of experiments were not considered. Instead, some other basic and relatively common experiments of this kind of works were run.

The main source of ideas for the selection of such experiments was the second paper presenting Taga's model [3]. Even though some experiments differ, it was considered as a good starting point for two main reasons:

- 1) Any experiment fitting properly on Taga's study must, therefore, fit properly in this study, as both are based on the same model.
- 2) This way it is easy to compare results between both studies.

### **Phase 6.- Writing the necessary code to plot the figures**

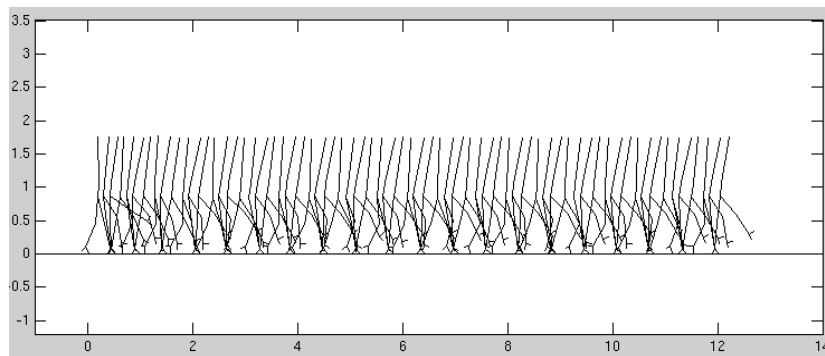
Two kinds of plotting were produced. On one hand, sets of arrays containing information about variables of interest such as forces or neural states were used to plot line charts. On the other hand, sets of arrays containing information about the position of each segment were used to plot diagrams of the robot similar to the one in Fig 3.3.

From now on, we will refer to these last diagrams as “stick figures”.

Several changes were inserted in the code to get the diagrams. The first kind of figures explained above were produced by plotting it in `step_man.py` using the function “`print`”. The idea was to insert the information in a file when the program was executed.

Plotting the stick figures was achieved by writing a new function “`dummy.py`” (in “`dummy`” folder) with the necessary code to plot the data on the console when the program is executed. This data is stored in a file which, later, is used as the input to the MATLAB function “`plotdummy.m`”, plotting the figure as a result. `Plotdummy.m` was created with this purpose, being compatible with the data produced by `dummy.py`.

For a small guide on how to produce stick figures, please refer to [annex A](#).



*Fig. 3.3: Stick figure of the robot's gait during steady walking*

## 3.2 Simulation environment

The space where the simulation will run is created in `model.py`, using the function “`space`”, which creates a class. Procedures within this class can create two different kinds of spaces: simple ones and hash ones. The difference between them are data structures used to manage the different geometries in the simulations and the algorithms used to calculate collision culling.

In our model we will use a simple space, in which collision culling is not present. The reason to choose this kind of space is that the number of present geometries is low, so the absence of collision culling does not slow the system down much. Also, not performing these calculations might avoid the presence of possible bugs related to the collision system.

Simulations are generated using the PyGame library (<http://pygame.org/>). The parameters of the simulation can be tuned in the `view.py` file. The most important parameters are:

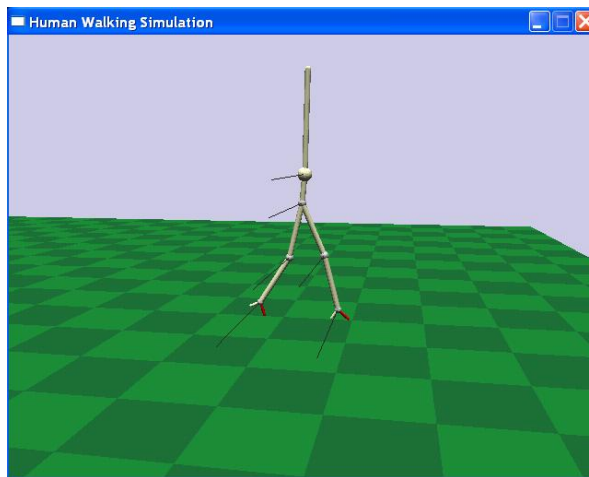
- “`rot`”: represents the rotation of the camera around the Z axis.

- “pitch”: represents the rotation of the camera around the X axis.
- “cameraDistance”: is the distance between the camera and the reference point.

The reference point is approximately linked to the robot and its coordinates are defined as follows:

- X coordinate: in the direction in which walking takes place. It is variable in time and equal to the coordinate of the HIP segment.
- Y coordinate: it is constant, as the movement is bi-dimensional.
- Z coordinate: it keeps a constant reference to the ground level at every moment.

The purpose of all these settings is a comfortable tracking of the system.



*Fig. 3.4: Simulation window*

### 3.3 The neural model

The CPG was first programmed in ANSI-C to minimize the computing time. The C-language was chosen because Python is capable of running C-code procedures. However, when the programming was finished, linking the code to use it in Python proved to be a long, complicated task.

As a result, code was translated into Python with the intention of linking the C-code later, after some successful simulations. However, after making some experiments changing the parameters of the simulation, it turned out that the neural network was not the most time consuming process. In fact, it was so quick to run that the code was altered, running several steps on the CPG for each step of the whole system. Because of this, the C-code procedures were never linked to the program.

Each oscillator in the CPG is constituted by two differential equations. These equations are an approximation to those used by Taga (see equations 2.1) using the Euler Method. They result in:

$$\begin{aligned}
u_i^{(N+1)} &= u_i^N + h/\tau_i[-u_i^N - \beta f(v_i^N) + \sum_{j=0}^{14} w_{ij}^0 f(u_j^N) + u_0 + Q_i + S_i], \\
v_i^{(N+1)} &= v_i^N + h/\tau_i[-v_i^N + f(u_i^N)], \quad (i=1,14) \\
&\text{where } f(u) = \max(0, u)
\end{aligned}
\tag{3.1}$$

Where  $u_i^N$  represents the state of the  $i$ th oscillator at the  $N$ th step and  $h$  is the time elapsed between two steps.

### 3.4 The robot

The robot was sized as the one in Taga's model except for the following:

- 1) Triangular feet were substituted by two segments with a fixed joint between them.
- 2) Restrictions for the movement in the knees were altered.
- 3) Links were formed by cylinders, instead of one-dimensional segments.

First, the constants and variables of the musculo-skeletal model are initialized and stored in a "robotModel" (file robot.py) class called "robot" at the initialization of model.py.

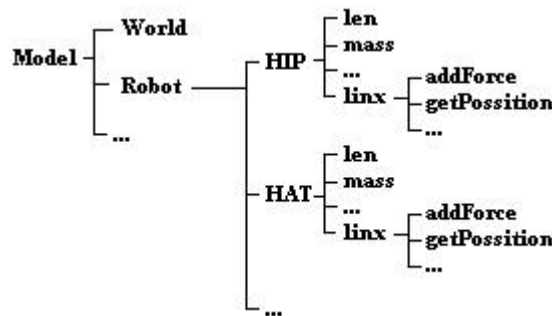


Fig. 3.5: Diagram showing the relations of different classes, variables and functions belonging to the class Model

#### 3.4.1 Segments

Segments are created in "robot.py", using the necessary data from bodyData.py and ICData.py. Each segment is associated to a class, stored in bodyData.py. Each of these classes contains the following data:

- length (.len)
- radius (.rad)
- mass (.mass)
- theta (.theta)

- angular velocity of theta (.thetaspeed)
- position (.x, .y, .z)
- forces (.force)
- torques (.torque)
- the extension .linx (explained bellow)

Some extra interesting data concerning robot definition is stored in ICDData.py. Each segment is defined as a different class here too, with each class sharing its name with the one storing data of the same segment in bodyData.py. Data stored in the classes contained in ICDData include:

- linear velocity (.linVel)
- angular velocity (.angVel)

The extension .linx is a class itself. It is a Body class from PyODE and contains the necessary information, concerning the segment that the physical engine needs. Because it is a Body class, it contains different functions and variables very useful for introducing changes in the segment at any moment, like “addForce” which lets us add a force to the existing ones in just one command or “getPosition” which outputs the current position of the segment as a triple.

### 3.4.2 Joints

The musculo-skeletal model presents the following kinds of joints:

- 1) Temporal joints
- 2) Permanent joints:
  - 2.1)Hinge joints
  - 2.2)Fixed joints

Temporal joints are created between the segments of the feet and the ground. They are necessary for the PyODE physics library to make a correct interpretation of the system. In case they were not created, feet would penetrate the ground. These joints are created every time step when they are necessary and they are destroyed after serving their purpose in that given time step, so they must be called multiple times in order to remain active. They are created at “model.py” when the library reports a collision between the ground and a segment.

In case the robot fell to the ground due to instability, temporal joints are also created between the ground and any segment in contact with it. This has the only purpose of making the simulation look more realistic.

Permanent joints of the robot are created at “robot.py”. These joints link the segments of the robot in an appropriate way. Two kinds of permanent joints exist in our model, depending on the number of degrees of freedom that they have. The joints connecting the segments of the feet are fixed, not offering any degree of freedom. The rest of the joints are hinge joints, which present one degree of freedom each. This is enough, as the system operates in two dimensions.

## 3.5 The ground

Ground is generated at model.py by the command “GeomPlane”, contained in ODE. This creates the ground as an infinite plane, perpendicular to the Z-axis. This plane is not completely drawn during the simulation as, at a given distance from the center, it apparently disappears. However, this is just a limitation of the graphical interface and the equations remain correct, so the robot does act as if the floor was still present.

## 3.6 Problems with first simulations (and solutions)

After finishing its programming, the code had to be reviewed several times while trying to run the simulation, in order to debug it. Little erratas in the equations were corrected too, but the robot kept on presenting two main problems we had to face:

### 3.6.1 The mechanical system had a high tendency to numerical instability

This kind of instability displays as if the model “exploded” in the simulation graphical interface. It is a common problem, according to documents dealing with it (<http://www.ode.org/ode-latest-userguide.html> 07-10-22).

There are a relatively large number of possible reasons leading to this problem like:

- stiff springs and forces
- large time step
- joints connecting large masses with small masses

This last point was in fact a problem, as feet (1kg each) where, by definition of the model, connected to shanks (3kg each) and their common joint had to deal with high forces and torques due to the weight of the whole body. In fact, numerical instability usually started in these joints. Nevertheless, these parameters could not be changed.

The other major influencing factors were tested and decreasing the time step proved to solve the problem, but at a cost of slowing down the simulation. Still, the need of such small time steps was not easily explained.

This new problem was not solved until some time later, by accident. During the experiments with up and down slopes, gravity was substituted in the code. Instead of using the function “setGravity” from PyODE library, a corresponding force was applied on each segment of the robot. From this moment, the presence of numerical instabilities was drastically reduced and the time step could be incremented noticeably (by more than ten times).



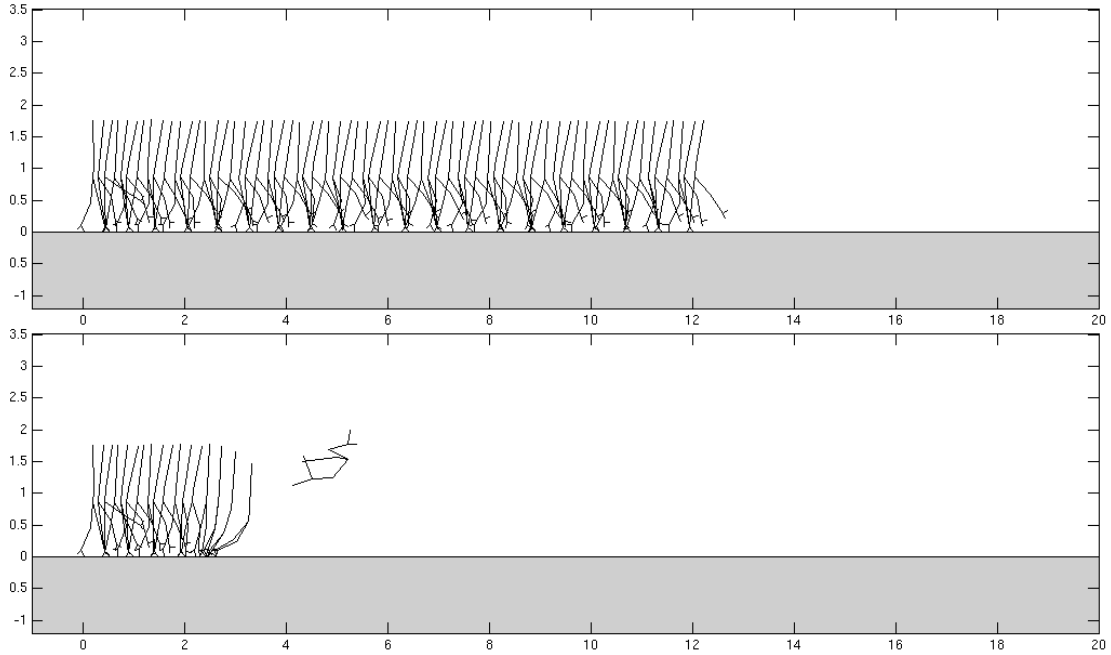


Fig. 3.6: Comparison between manually setting the corresponding forces (up) and using the “setGravity” function from PyODE (down)

Later on, a new change was introduced in order to increase numerical stability. Angular velocities were corrected as shown in the following formula:

$$\dot{\theta}_t[i] = 0.9 * \dot{\theta}_{(t-1)}[i] + 0.1 * \dot{\theta}_t[i] \quad (3.2)$$

It is possible to generate steady walking without this correction. Nevertheless, stability increases in general when it is included.

### 3.6.2 The system was unstable and, after a couple of steps, it fell to the ground

The second problem could have been attributed to a difference in the environment, and a possible way of fixing it would have been to re-tune the weights of the neural system. This would have proved to be a high time consuming process indeed, so we decided to try other routes before.

Forces were applied in the foot in contact with the ground during swing phase and a force was applied in the left leg during the first seconds of the simulation in a try to bring the system to a more stable configuration.

We hoped the system to be able to achieve some more steps, but after some tries a configuration was reached from which the system proved to be indefinitely stable.

The explanation for this is that the system is very sensible to the starting point configuration.

## **3.7 Corrections**

Some corrections had to be introduced so the system was able to reach stable gait. Theoretically, this would not be necessary. But there are differences between our system and the one used by Taga, even though they could seem small, are large enough to take the system out of the limit cycle.

### **3.7.1 Correcting force**

It is difficult to find a set of positions and velocities of the segments that, together with the starting conditions of the neural system, are contained within the limit cycle. The reason for this is that the system has a high number of degrees of freedom. Therefore, there are many related variables that should be tuned together to reach this goal.

Instead, starting from a good approach to this solution (as could be the one proposed by Taga), it is logical to think that the system will be close to the limit cycle. The reproduction of graphical simulations of the system's behavior can then lead the developer to guess possible corrections. The application of these corrections may bring the system to points within the limit cycle.

A number of correcting forces were tried until the system started to be able to reach and maintain stability. The best solution that could be found was a force of 1118N applied to the left shank during 65ms, at the start of the simulation.

### **3.7.2 Linkage of feet with the ground**

The application of forces may lead to points closer to the limit cycle. But, even though it could theoretically be reached, this solution would be extremely sensible to slight changes in parameters of the system like those in slopes and gravity.

A way of diminishing this problem is to increase the attractor zone of the limit cycle. This was done by adding forces on the feet every time there was a risk for both of them not to be in contact with the ground. This situation is undesirable for several reasons:

- The system is designed for walking, not running. During walking there is always, at least, one foot in contact with the ground. Therefore, this situation is not predicted by the model. In fact none of the six states would be reached as none of their conditions would be fulfilled.

- Torques applied in the joints are too big in case the robot is not in contact with the ground, as they are not transmitted to it. This leads to big accelerations and, often, numerical instabilities.

During swing phase, a force was applied at the foot in contact with the ground until the limit cycle was reached. This moment is not easy to determinate, so it was assumed to take 1.3s to reach.

## 4.- Results

### 4.1 Steady state

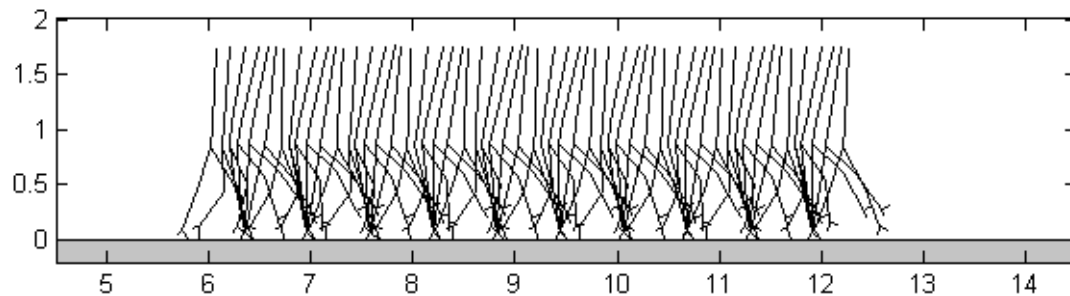


Fig. 4.1: Stick figure of gait in steady-state walking, traced every 0.09s

The very first objective of this project was to produce stable gait with our model. This was not an easy task, as the influence of parameters such as the time-step is crucial and it was unknown. The robot was, also, extremely sensitive to initial conditions. Therefore, they had to be carefully adapted.

The best results were obtained with a time step of 0.13ms. The starting position and velocities of each segment were the same as those presented in Taga's paper [2]. Nevertheless, some corrections had to be made, as was explained in chapter 3.7.

#### 4.1.2 Simulation results

In this section we will cover steady walking and, also, the progress since the start of the simulation until steady walking is established. The whole simulation is run for 10s and results at this section will normally correspond to this same simulation time. An exception are those experiments showing results for one gait cycle.

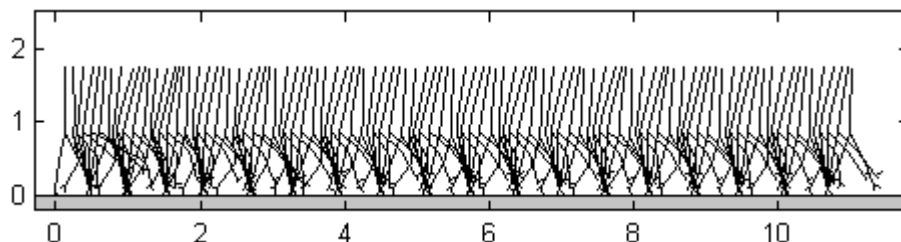


Fig. 4.2: Stick figure of steady gait on a flat surface, traced every 0.09s. Scales refer to distances in meters

Taking a look at the obtained stick figure (Fig. 4.2), we see that gait is very similar to human walking. Nevertheless, obvious differences exist because of the big simplifications present in the model. First, the model is two-dimensional so results should be interpreted as a projection of a theoretical human gait on a plane.

Another clear lack of the model is the absence of a hip, which leads to both legs starting at the same point. Instead, a more realistic simulation of human walking should show a relative movement of the heads of both femurs even at the projection of gait that we are considering.

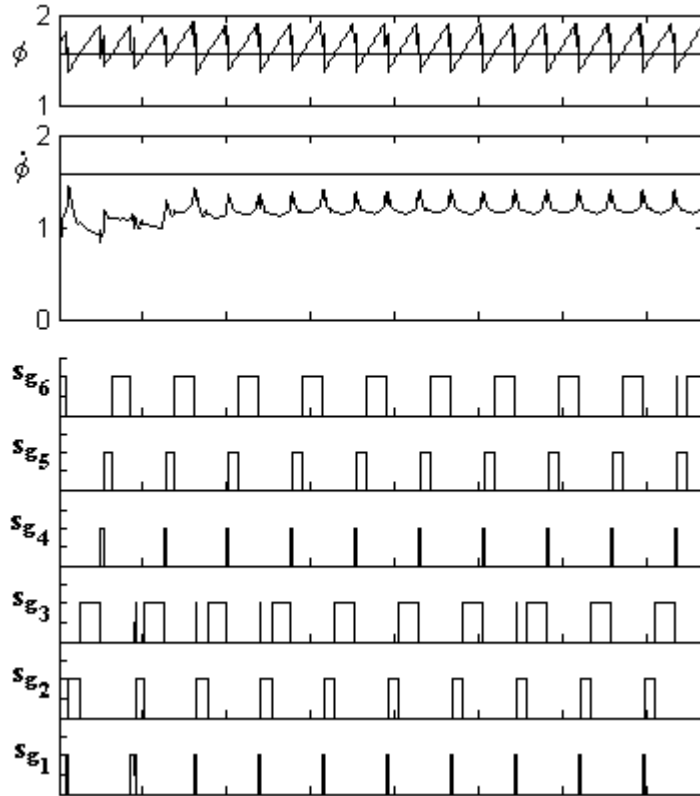


Fig. 4.3: Global angle, global angle velocity and activity of each of the possible states of the system during establishment and maintenance of steady walking gait

Global angle shows its typical oscillatory behavior. A closer look at the diagram (Fig. 4.3) shows that each cycle is composed of a progressive increase followed by a sudden decrease. To explain this we must take a look at its definition

$$\Phi = \cos^{-1}[(x_{COP} - x_{COG}) / ((x_{COP} - x_{COG})^2 + (y_{COP} - y_{COG})^2)] \quad (4.1)$$

where COP and COG refer to the center of pressure and the center of gravity of the robot, respectively. The reason why decreases of global angle occur faster than

increases is that double stance (decrease) takes a short time and, while it does, COP moves from the back foot to the front one.

A closer look on the diagram shows read errors of the global angle. They appear because of bounces of the feet at the moment swing ends and the foot strikes the ground.

Global angular velocity starts in a value defined by the initial conditions, increases, and reaches a stable oscillatory behavior. This matches with the results present at Taga's paper, with an exception: his results show a noticeably smoother stabilization process. This difference is, probably, due to the difficulties that our robot suffers in order to reach the limit cycle (including the application of an external force), while Taga's robot starting conditions are included in it.

Results of global state show how the robot starts the simulation with both feet on the ground and an important inertia that immediately makes it enter a swing phase. Looking at the diagram, we can notice how the system passes through its different states in order, with the exception of some transitions, like the one from sg1 to sg2 in which it jumps to sg3 for a number of steps. This is due to read errors of the global angle (discussed before). As long as they are short, they should not pose a problem but, if they are long enough, they can turn the system unstable.

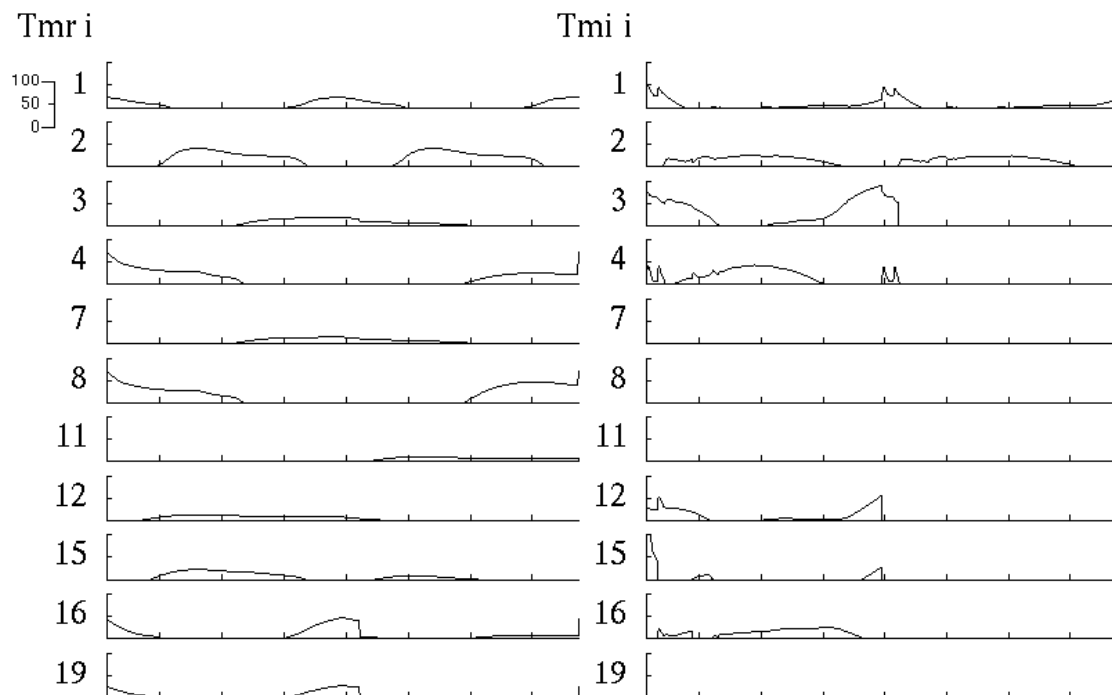
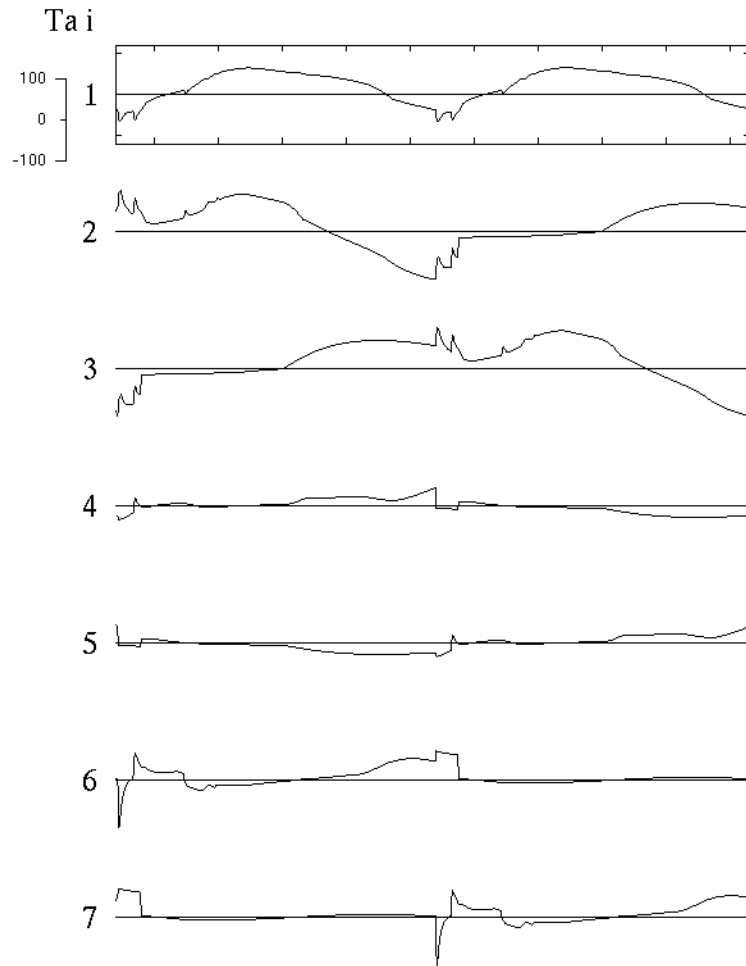


Fig. 4.4: Torques originated by the neural network (left) and torques originated by the impedance controller (right) during one steady walking cycle. Vertical scaling refers to torques in  $N*m$ . Some torques were not included because of symmetry with present ones (for example, 10 is symmetric to 8)

Torques originated by the Neural Rhythm Generator were next to analyze. Torques of muscles 3 to 20 match with those of Taga, but torques number 1 and 2 present a phase difference of  $\pi/2$ . Equations were checked several times, trying to explain this, but no errors were found. One possible explanation is that the system might have reached a

limit cycle different to the one of Taga. This does not look probable, though, as stick figure diagrams show similar gaits. Another explanation of this would be a misprint at Taga's paper.

Torques originated by the Impedance Controller match with Taga's ones. This is true even for those corresponding to rectus abdominis (number 1) and erector spinae (number 2), which reinforces the theory of the misprint explained before.



*Fig. 4.5: Torques (in  $N*m$ ) at each of the seven joints during a steady gait cycle. The grid has not been removed at the first one to facilitate measures. Presented torques correspond to: 1) abdomen 2) right hip 3) left hip 4) right knee 5) left knee 6) right ankle 7) left ankle*

A diagram of the active joint torques generated by the muscles is presented in Fig. 4.5. Results are very similar to the ones obtained by Taga. We can see how torques of symmetrical joints are the same, with a phase difference of  $\pi$ .

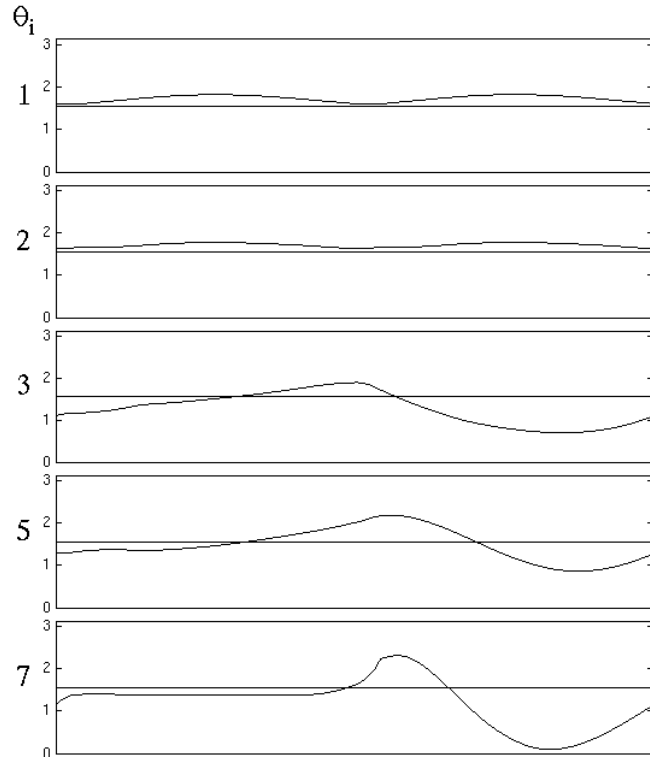


Fig. 4.6: Angles of the segments of the robot during a cycle of steady gait. Horizontal lines corresponding to  $\pi/2$  are drawn to serve as a reference. Angles correspond to: 1) HAT 2) pelvis 3) thigh 5) shank 7) foot

If we analyze the diagram of the evolution of the angles (Fig. 4.6) we find out that they perfectly match with Taga's diagram, except for the foot. Foot's diagram is vertically displaced compared with the one by Taga. The reason for this to happen is that, as the position of the COM of the feet was not defined in Taga's paper [2], a reference for the angle of the feet had to be estimated. As a result, displacement in this diagram is equal to the error made in the estimation.



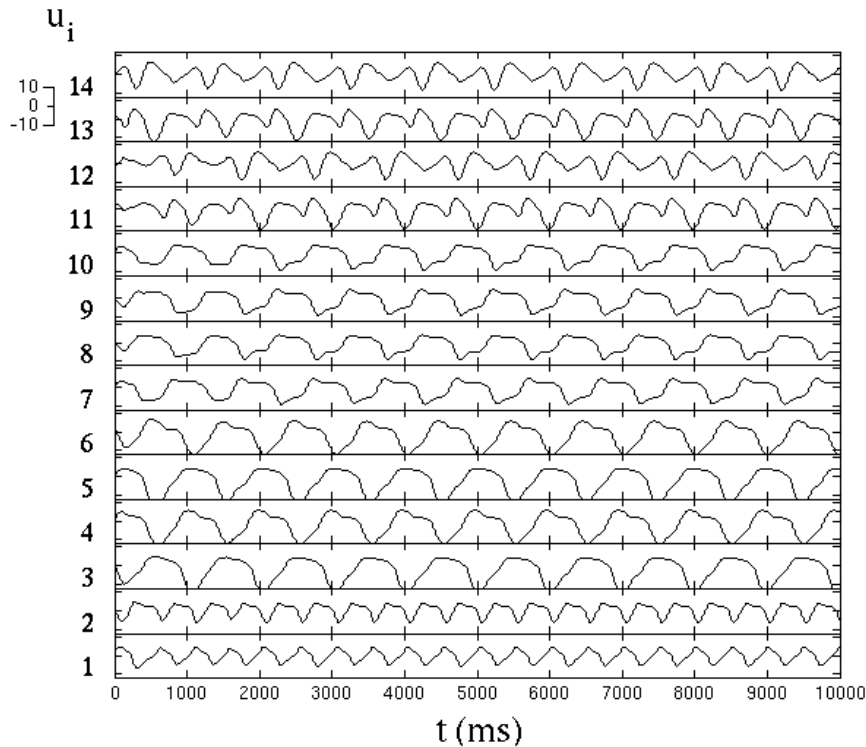


Fig. 4.7: Output of the 14 neurons during our simulation of establishment and maintenance of steady gait

One last interesting set of data that was analyzed is the one formed by the output of the Neural Rhythm Generator. These values directly affect the torques exerted by muscles and, also, constitute the internal state of the neural system. Results, once stabilization is achieved, are very similar to those obtained by Taga. It is interesting to note how neurons 1 and 2, which define the movement of the abdomen, oscillate twice as fast as the rest of the neurons.

## 4.2 Simulating slopes

This set of experiments is interesting for three reasons:

- 1) It evaluates the changes produced in stable gait when slope is different from zero.
- 2) It allows us to find the maximum and minimum slopes that may be reached, with the system being able to achieve stable gait.
- 3) It allows us to find out and analyze the way the robot falls to the ground when stable gait may not be achieved.

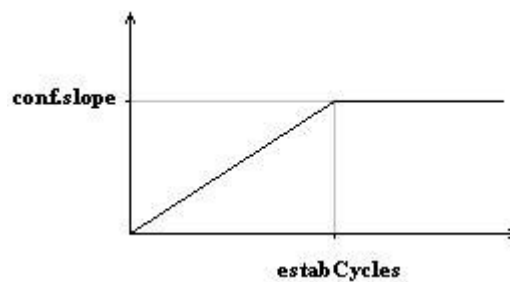
The slope is altered by directly affecting the force of gravity in each segment of the robot. The gravity force applied on the  $i$ th segment will be:

$$Fg=(m \cdot g \cdot \sin(\theta), 0, m \cdot g \cdot \cos(\theta)) \quad (4.2)$$

Gravity is changed by function “changeGrav” in bodyData.py, with “thetaWorld” ( $\theta$ ) as its only input. After considering several options to simulate slopes, we decided to take two different sets of experiments:

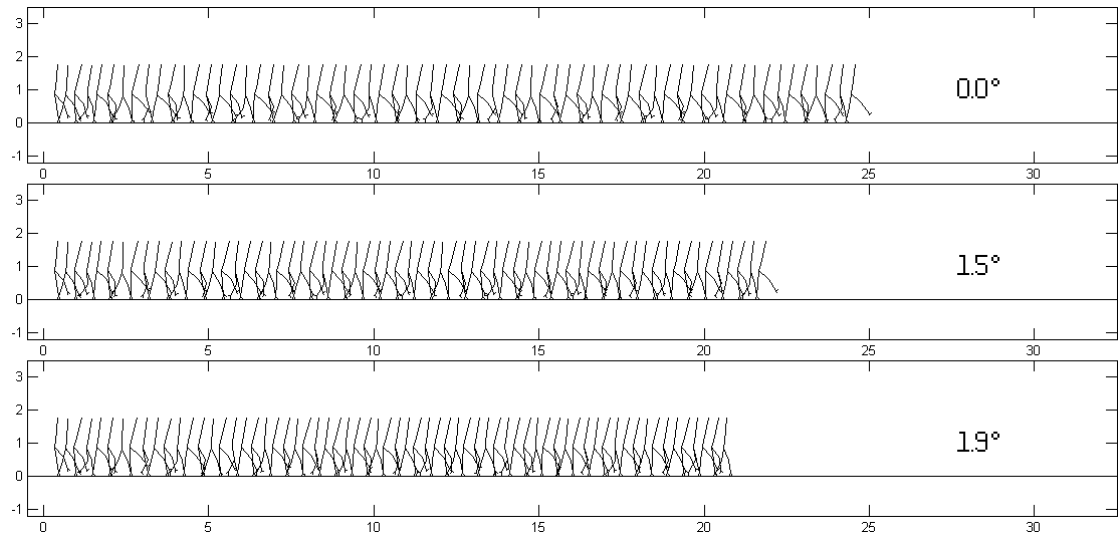
### 4.2.1 Progressive slope change

Starting from zero, the slope is progressively increased or decreased until the objective slope is reached and, from here on, it remains constant.

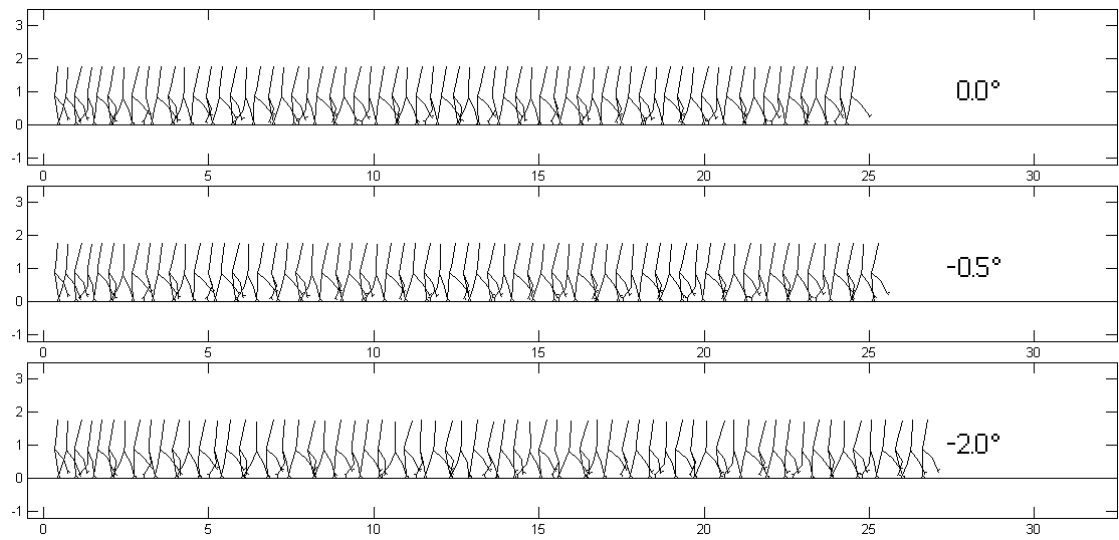


*Fig. 4.8: Slope evolution during a progressive increase slope test simulation*

The reason to gradually increase the slope is to allow the system to reach the limit cycle without changing the initial conditions. Also, letting the oscillator attractor to work during a longer time, the system can be smoothly brought through new limit cycles. This way, for a long enough time of stabilization, instabilities would be always caused by the system being unable to perform at the slope being tested. On the other hand, for too short stabilization times, it wouldn't be clear if the reason is this or if instabilities are due to the attractor not being fast enough to adapt to slope changes.



*Fig. 4.9: Stick figures of the robot traced every 0.3s during 20s with different ascent slopes. Stick figure of steady walking was included to facilitate comparison. The angles written on the right correspond to the maximum slopes programmed for the experiment. Gradual increase of slopes take 5s in all figures. Scaling in meters*



*Fig. 4.10: Stick figures of the robot traced every 0.3s during 20s with different descent slopes. Stick figure of steady walking was included to facilitate comparison. The angles written on the right correspond to the maximum slopes programmed for the experiment. Gradual increase of slopes take 5s in all figures. Scaling in meters*

## 4.2.2 Sudden slope change

This set of experiments tries to evaluate the ability of the system to react against one kind of perturbation: the sudden change of slope, whether it is a sudden increase or a sudden decrease.

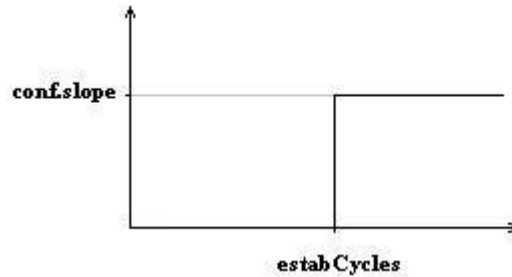


Fig. 4.11: Slope evolution during a sudden increase slope test simulation

Results presented in the following figures (Fig. 4.12 and Fig. 4.13) show the performance of the system at the higher sudden slope increase and decrease that could be achieved with the system not becoming unstable. The robot decreases its speed in downslope and increases it in upslope, as was expected. Another consequence of slope change is an alteration of the amplitude of the oscillations of the global angle. These results are similar to those obtained by Taga.

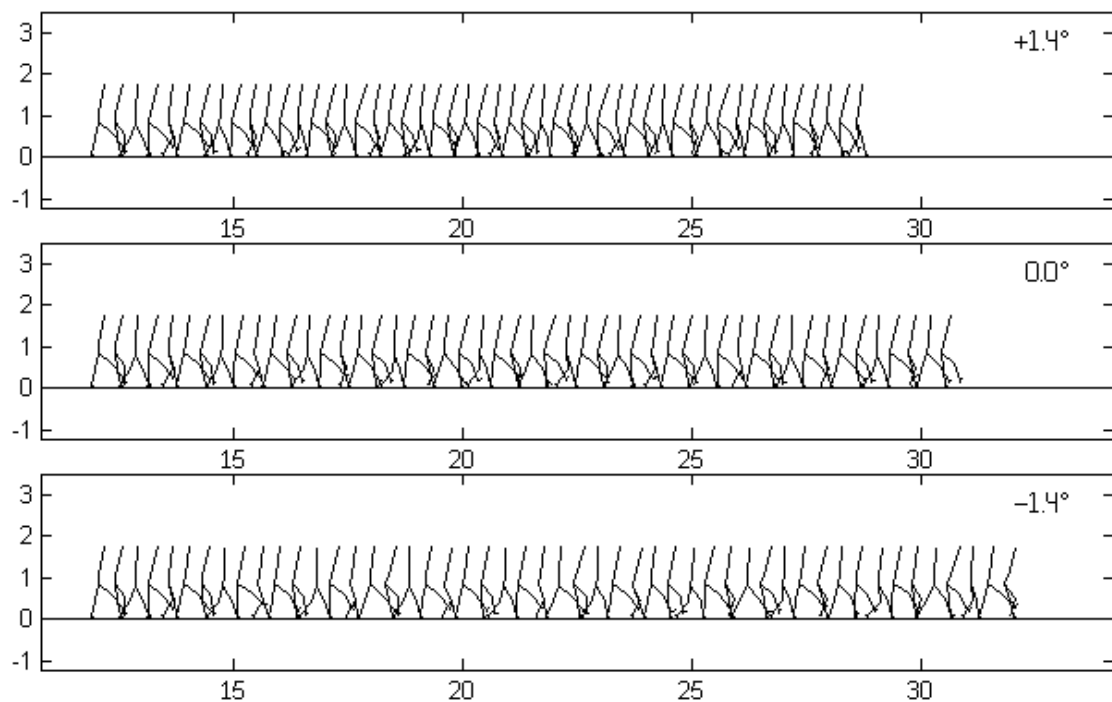


Fig. 4.12: Stick figures showing the evolution of gait when the robot faces several sudden slope changes at  $x = 14$ . Scaling in meters

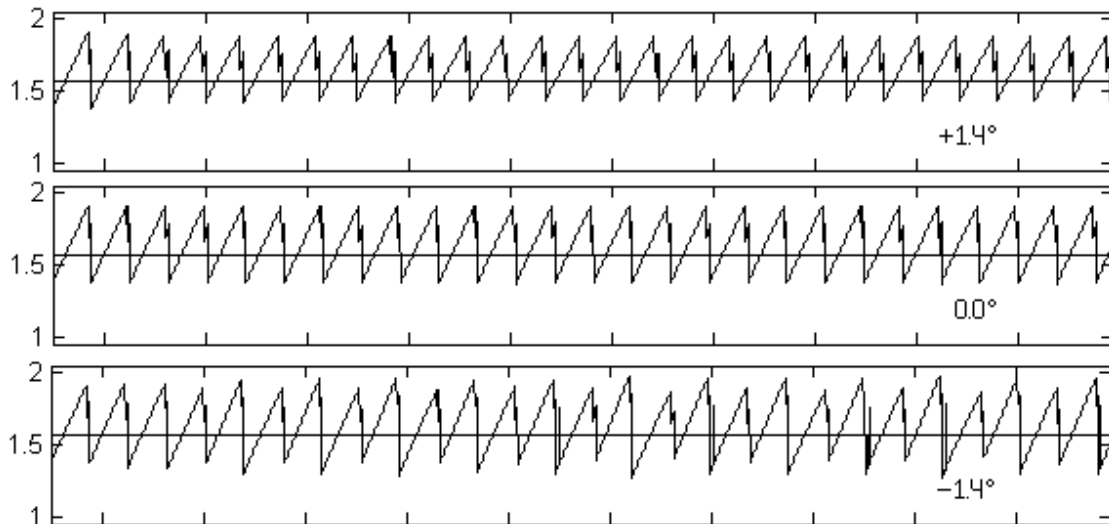


Fig. 4.13: Evolution of the global angle when the robot faces several sudden slope changes at  $x = 14$

### 4.3 Gravity changes

The system was subjected to several experiments changing the value of the gravity. These experiments had two goals:

- 1) To test how the system reacted, checking if it adapted its gait to the new situations.
- 2) To find the limits of gravity where the system could achieve stable gait.

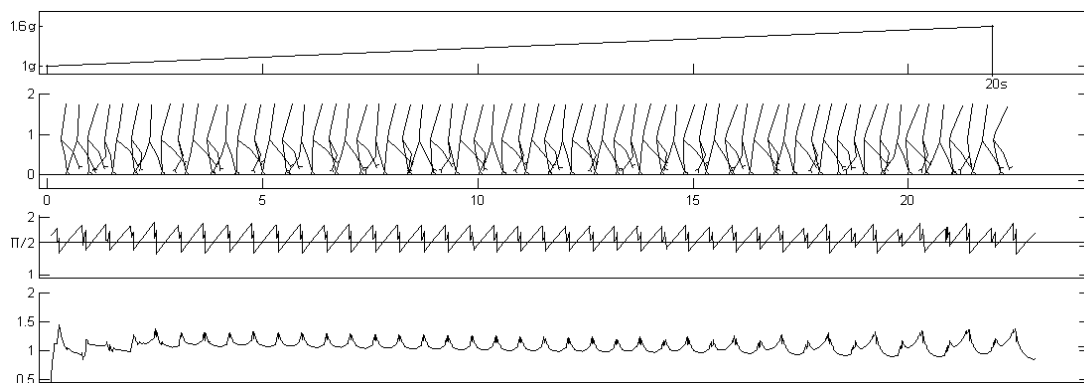


Fig. 4.14: Evolution of gravity. Stick figure traced every 0.3s (scaling in meters). Global angle. Global angle's velocity

In order to find the limits of gravity where the system kept stability, forces of gravity were gradually increased. The reasons for changes to be gradual are similar to those explained before, in gradual slope changes.

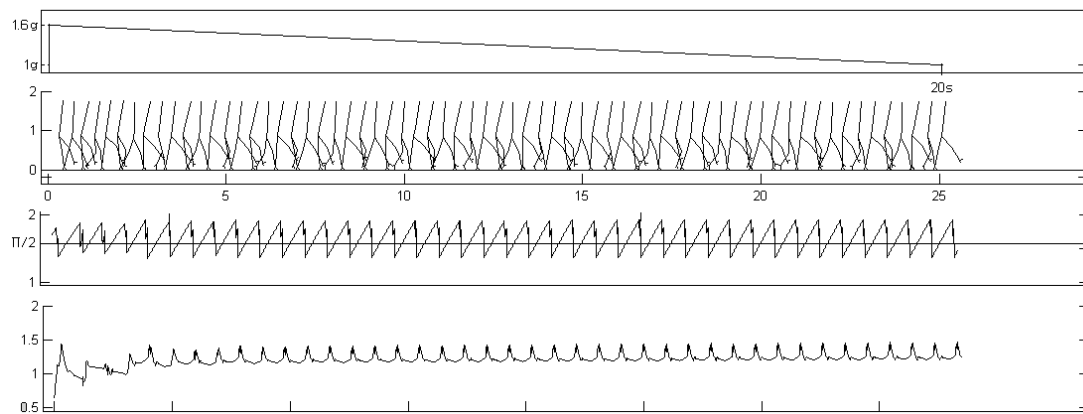


Fig. 4.15: Evolution of gravity. Stick figure traced every 0.3s (scaling in meters). Global angle. Global angle's velocity

The system is able to maintain stable gait when submitted to small changes of gravity. Different experiments concluded that, in general, less gradual changes in gravity lead to smaller times for the system keeping stability.

Small changes of gravity are unable to turn the system unstable even if they are maintained for long periods of time. It seems that these periods of time could have infinite length for gravities between two given (unknown) values. Note that this could, or not, include  $g = 9.81 \text{ m/s}^2$ , even though it probably would.

## 4.4 Perturbations

Once stable gait is successfully achieved, the system is supposed to maintain it under small enough perturbations, due to its oscillator nature. Naturally, an important question here is how small these perturbations must be before they turn the system unstable. Unfortunately, this is not easy to answer, as many different kinds of perturbations may be applied at any given time during a gait cycle, producing many different reactions.

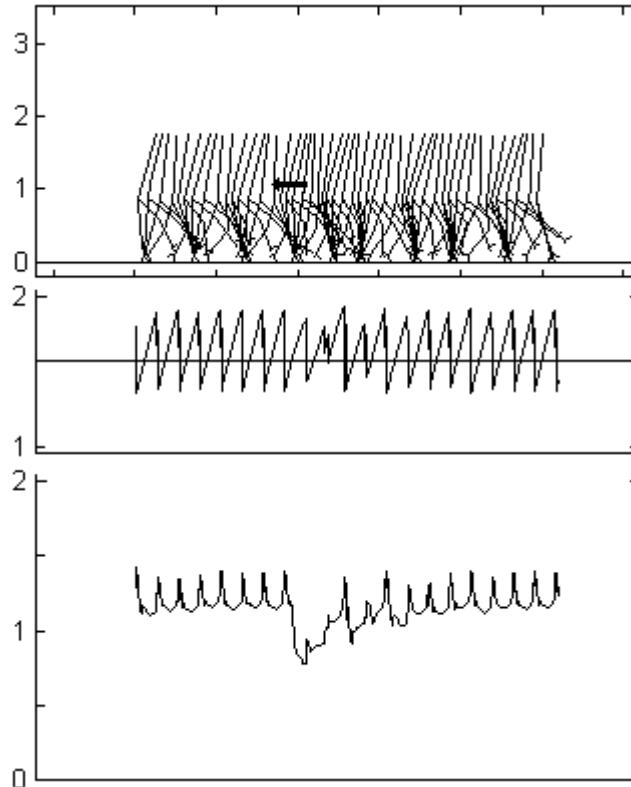
As a result, it was necessary to refer to those experiments present at Taga's paper [3]. The purpose of these experiments is to compare the results with those from Taga's paper to evaluate the resemblance between them. The experiments are:

### 4.4.1 Sudden short perturbation

A force of 200N was applied in the center of mass of the HAT in walking direction

and opposite way during 0.1s. The result of this experiment is shown in Fig. 4.16. When it is compared with the one by Taga, the following conclusions are drawn:

- The gait is affected in a similar way.
- In both cases, the global angle is not allowed to progress for a short period of time and, when released, its amplitude increases.



*Fig. 4.16: Up: Stick figure of gait with a force of 200N applied during 0.1s at the HAT as indicated. Figure was traced every 0.09s from  $x = 5.0$  to  $x = 10.0$ . Scaling in meters keeping proportions. Center: Evolution of the global angle. Down: Evolution of the global angle's velocity*

#### 4.4.2 Sudden increase of weight

Weight of pelvis was suddenly increased by 15kg (147.15N) during steady walking. Results of this experiment are shown in Fig. 4.17. The following similarities were found when they were compared with those of Taga:

- There are no noticeable changes in gait on a first look on both stick models.
- The amplitude of the global angle decreases and so does the average global angle velocity.
- The amplitude of the global angle velocity remains, approximately, the same.

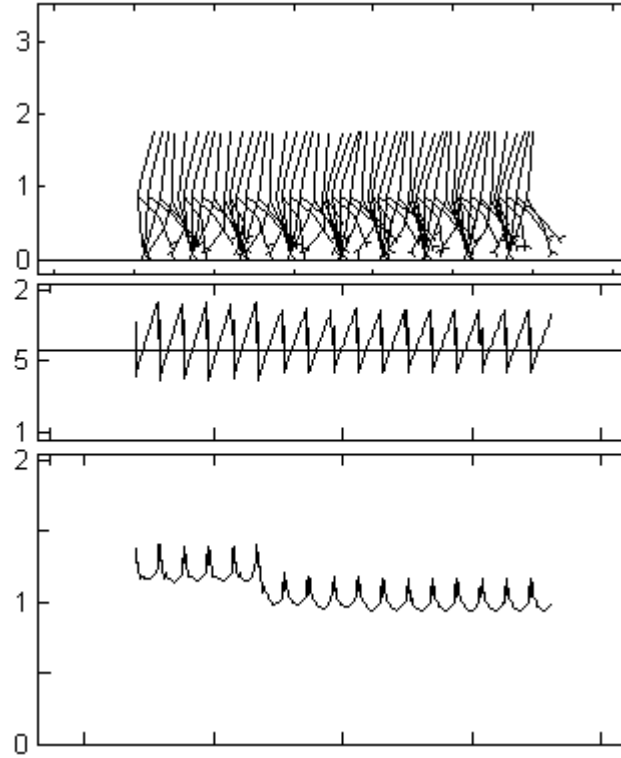


Fig. 4.17: Up: Stick figure of gait with a force of 15kg applied from a given moment at the HIP (traced every 0.09s from  $x = 5.0$  to  $x = 10.0$ ). Scaling in meters keeping proportions. Center: Progression of the global angle. Down: Progression of the global angle's velocity

## 4.5 Importance of the global angle

As the code was written, the global angle was first approximated by a simple relation between the position of both feet. This was done thinking that the result could be similar enough to produce short simulations while developing other areas of the system first. The results were not able to show any stable wait, even though this could be due to some other reasons.

The global angle was introduced in one of the final stages of the programming, producing an improvement of the gait. After having developed a system capable of reaching stable gait we wondered how important the global angle really was.

In the beginning, a series of experiments were run on the system changing the definition of the states. This time, instead of using the global angle, just the relative positions of the feet among themselves and the ground were used. The old definitions are presented, followed by the new ones.

$$\begin{aligned}
 s_{g1} &= s_{ron} s_{lon} s_r, & s_{g2} &= s_{ron} s_{loff} 1(\pi/2 - \Phi), & s_{g3} &= s_{ron} s_{loff} 1(\Phi - \pi/2) \\
 s_{g4} &= s_{lon} s_{ron} s_l, & s_{g5} &= s_{lon} s_{roff} 1(\pi/2 - \Phi), & s_{g6} &= s_{lon} s_{roff} 1(\Phi - \pi/2)
 \end{aligned} \quad (4.3)$$



$$\begin{aligned}
s_{g1} &= s_{ron} s_{lon} s_r, & s_{g2} &= s_{ron} s_{loff} s_r, & s_{g3} &= s_{ron} s_{loff} s_l \\
s_{g4} &= s_{lon} s_{ron} s_l, & s_{g5} &= s_{lon} s_{roff} s_l, & s_{g6} &= s_{lon} s_{roff} s_r \quad (4.4)
\end{aligned}$$

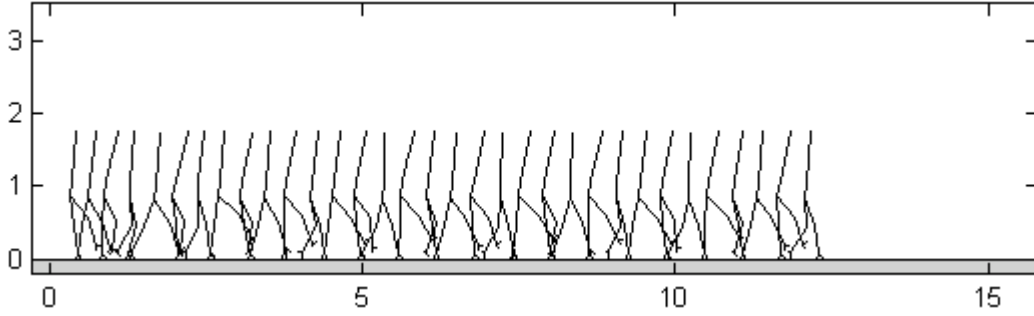


Fig. 4.18: Stick figure of steady walking with global states independent of global angle. Scaling in meters

Results show that the system is able to reach and maintain stable gait as shown in Fig. 4.18. Nevertheless, the global angle does not only affect the global states. It is also used in the sensory input for the neural system. So, as a second step, the global influence of the global angle was eliminated from this too.

The global angle was substituted by a function just dependent on the global states. Several simulations were run to find the average value of the global angle during each of the states of the system for steady walking.

$$\begin{aligned}
s_{g1} = 1 &\rightarrow \Phi = 1.7121 \\
s_{g2} = 1 &\rightarrow \Phi = 1.4727 \\
s_{g3} = 1 &\rightarrow \Phi = 1.7429 \\
s_{g4} = 1 &\rightarrow \Phi = 1.7265 \\
s_{g5} = 1 &\rightarrow \Phi = 1.4750 \\
s_{g6} = 1 &\rightarrow \Phi = 1.7439
\end{aligned}$$

In theory, global angle should evolve similarly in symmetric states (1 and 4, 2 and 5, 3 and 6), so data was corrected finding the mean of these states.

$$\begin{aligned}
s_{g1} = s_{g4} = 1 &\rightarrow \Phi = 1.7193 \\
s_{g2} = s_{g5} = 1 &\rightarrow \Phi = 1.4738 \\
s_{g3} = s_{g6} = 1 &\rightarrow \Phi = 1.7434
\end{aligned}$$

Our last step would be dealing with the global angular velocity, which influences the sensory input for the neural system. It was treated the same way as the global angle, getting the following results:

$$s_{g1} = s_{g4} = 1 \rightarrow \dot{\Phi} = 1.3358$$

$$s_{g2} = s_{g5} = 1 \rightarrow \dot{\Phi} = 1.2488$$

$$s_{g3} = s_{g6} = 1 \rightarrow \dot{\Phi} = 1.1757$$

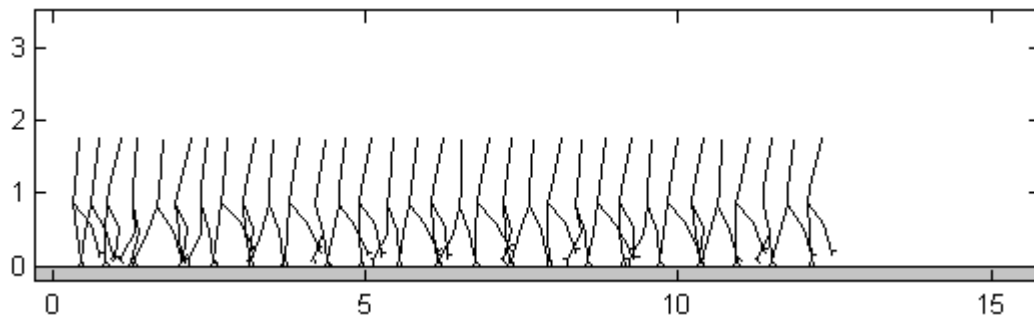


Fig. 4.19: Stick figure of steady walking with global states and neural input independent of global angle and global angular velocity during 10s. Figure traced every 0.3s. Corrections on the sensory input are applied from  $t = 2s$ . Scaling in meters

The corrections on the sensory input could not be applied from the beginning of the simulation because the system was unable to reach stability without changing the initial conditions. It seems logical to think that, changing these conditions, stable walking could be reached applying the changes from the start.

## 4.6 Changes in time step

When the time elapsed between two re-calculations of the system parameters is small enough, further decreases should result in little changes, besides a higher computation time. This did not prove true during the development of the system, so some simulations were run in order to observe to what extent this was true.

Four experiments are presented (Fig. 4.20) in which the robot evolves from the same initial conditions without any slope. The only difference is the step time.

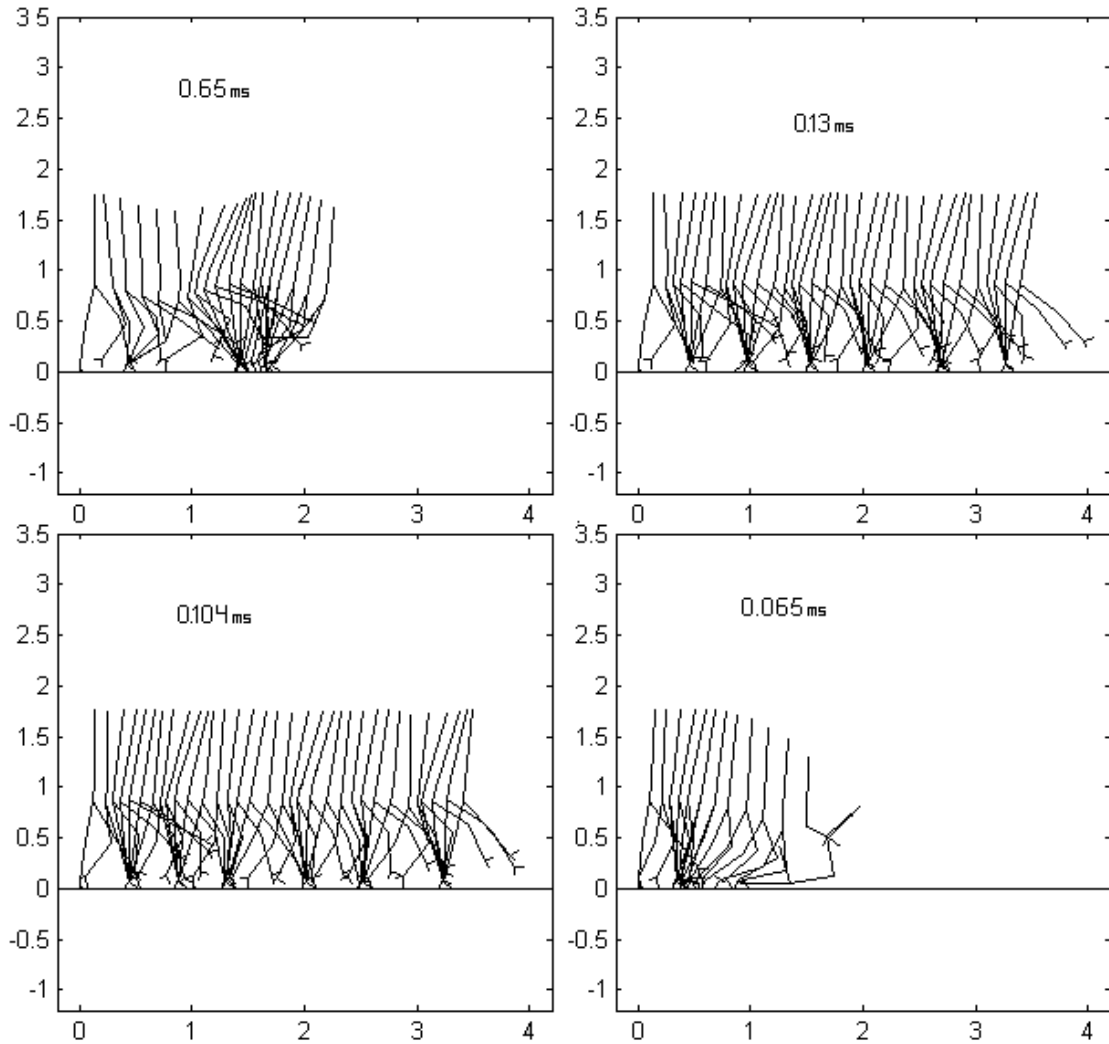


Fig. 4.20: Stick figures for the evolution of the robot with different values of time step. Scaling in meters

If the step time is further decreased, the influence of gravity disappears and the robot starts “floating”. This could be due to rounding in operations, as the influence of forces is proportional to the time-step in ODE engine.

A compromise solution must be reached for the time-step. To generate realistic simulations, it must be small enough not to produce big errors and big enough not to ignore too many forces because of rounding. The band where both objectives are fulfilled is narrow enough not to consider further objectives, such as producing fast simulations to save time in getting results.

## 4.7 Robustness of the model

In order to check how sensitive the system was to different parameters, a new set of experiments were run altering their values and checking if the system was able to maintain a stable gait.

Due to the nature of these experiments, changes take place after steady walking has been reached. Changes happen suddenly, which makes the system more sensitive to them than it would if they were applied in a progressive way.

Results were compared with those obtained by Taga. Taga's paper [3] did not specify if changes were gradually applied, so we will assume they were not.

### 4.7.1 Changes in the strength of neural connections

The strength of the neural connection between the  $i$ th and the  $j$ th neuron is noted as  $w[i][j]$  and is stored in a symmetric 14x14 matrix. It is not written in Taga's paper if changes were applied in all the neural connections but it seems probable.

Results: -11% - +11%

Taga's Results: -25% - +100%

There is a big difference between these results. The reason for this to happen could be that Taga changed the weights of just some of the neural connections and not all of them, as we did. Still, this is not specified in his paper, so even if he did, it would still be unknown which were changed and which were not.

It seems unlikely that his results could correspond to the same experiment that we have run, so no further conclusion was extracted from this experiment. It is presented here because it belongs to the same set of experiments as the following ones, in Taga's paper[3]. Also, it might be useful as a comparison point for future experiments.

### 4.7.2 Changes in sensory inputs

Vector  $S$ , composed of 14 scalars, was multiplied by a constant factor,  $F$ , from  $t = 3s$ . Evolution was recorded.

$$S[i] = S[i] * F \\ i \in (1, 14)$$

The purpose was to find the limits of  $F$  where the system is stable, but during the experiments an interesting behavior was shown too. Within the stability limits, the

average speed of the robot increases with  $F$ . The explanation of this may be that an increase in the value of  $F$  results in an increase in the average torques applied in the joints. In fact, several other experiments increasing this average value increase the average speed of the robot, to a greater or lesser extent, too.

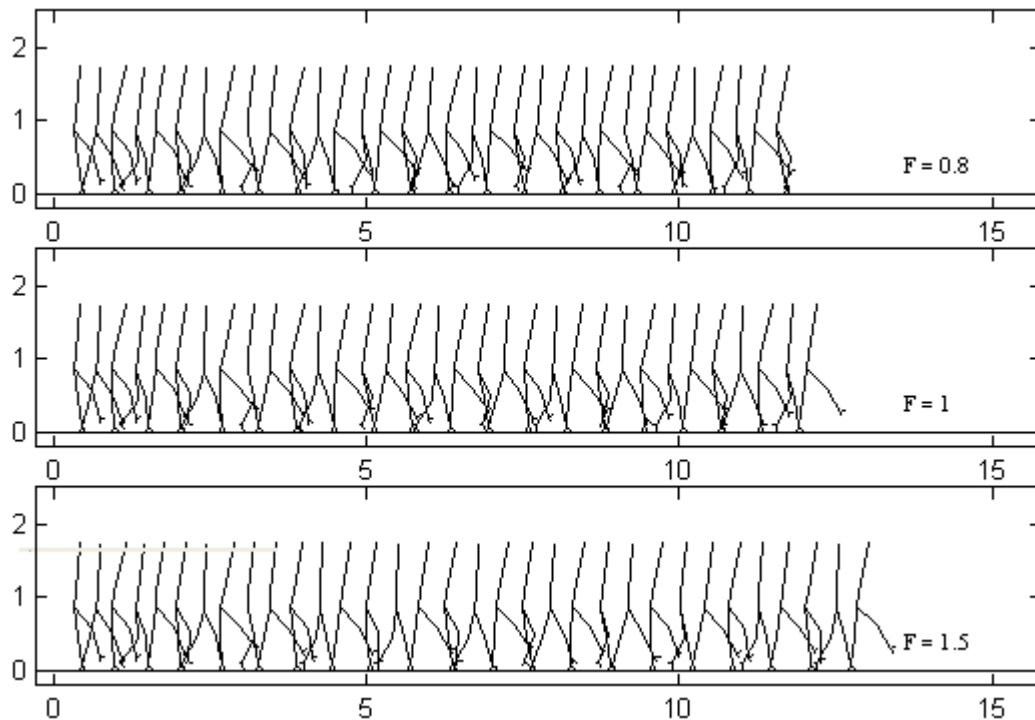


Fig. 4.21: Stick figures of the robot during stable gait for different values of  $F$ . Scaling in meters

The limits of  $F$  were found as:

Results: -20% - +50%

Taga's Results: -15% - +35%

The range offered by the results is similar to the one found at Taga's paper.

### 4.7.3 Changes in impedance parameters

Multiplying all of the impedance controller's parameters by a given factor is equivalent to multiply the output of the controller by that same factor. The purpose of these experiments is to understand how dependent of a fine tuning of the impedance controller the system is.

Results: -40% - +20%

Taga's Results: -20% - +50%

Even if the results seem quite different, the ranges are similar (60% and 70%). The conclusion is that the system is reasonably stable when submitted to changes in these

parameters.

#### 4.7.4 Changes in Rhythmic Force Controller's parameters

The result of multiplying the coefficients of the Rhythmic Force Controller ( $p_1, p_{18}$ ) by a given value, is multiplying the muscle torques generated by the neural controller ( $T_{mr1}, T_{mr20}$ ) by the same value.

Results: -50% - +15%

Taga's Results: -10% - +10%

Results here are specially good when we compare them with the ones of Taga. It seems strange that the range of possible values is not centered in the default ones. This may be caused by the effect that rounding has on gravity, slightly decreasing its value with the used time step (0.13ms).

#### 4.7.5 Changes in neurons' time constants

$\tau_i$  and  $\tilde{\tau}_i$  are time constants defining the adaptation capacity of the  $i$ th neuron. Bigger values lead to slower transitions and lower values lead to faster ones.

Results: -5% - +10%

Taga's Results: -10% - +10%

As expected, ranges are similar. The system proves very sensitive to changes in the values of these parameters as they must be well tuned for the system to follow a coordinated progression of its states.

## 5.- Conclusion

In this paper we have tried to answer the question “is PyODE a useful tool to use if you want to simulate biped walking?”. This is, of course, a question that can not be answered with a simple statement. PyODE offers good features for this task, like simplicity, easiness to debug, a reasonably good documentation and being an open-source free library. Some drawbacks were discovered during the experiments that were carried out in this investigation, on the other hand. It has some bugs and a high tendency to numerical instability which may difficult or even impede its use in some situations. Even though, the performance of the system was impressive in some ways. When its results were compared with the ones obtained by Taga, the system proved to be very resistant to perturbations. This information may be useful for future work about human gait and even studies dealing with other kinds of biped walkers, inverted pendulums and other related areas.

## 6.- References

- [1] “A combined neuronal and mechanical model of fish swimming” Ö. Ekeberg. *Biol. Cybern.* 69 (1993) 363-374
- [2] “A model of the neuro-musculo-skeletal system for human locomotion. I. Emergence of basic gait” G. Taga. *Biol. Cybern.* 73 (1995) 97-111
- [3] “A model of the neuro-musculo-skeletal system for human locomotion. II. Real-time adaptability under various constraints” G. Taga. *Biol. Cybern.* 73 (1995) 113-121
- [4] “A Reflexive Neural Network for Dynamic Biped Walking Control” T. Geng, B. Porr, F. Wörgötter. *Neural Computation* 18 (2006) 1156-1196
- [5] “An alternative approach to synthesizing bipedal walking” H. van der Kooij, R. Jacobs, B. Koopman, F. van der Helm. *Biol. Cybern.* 88 (2003) 46-59
- [6] “An analysis of neural models for walking control” R. Reeve, J. Hallam. *IEEE Transactions on Neural Networks*, Vol 16 (2005) 733-742
- [7] “An Empirical Exploration of a Neural Oscillator for Biped Locomotion Control” G. Endo, J. Morimoto, J. Nakanishi, G. Cheng. *Proceedings of the 2004 IEEE International Conference on Robotics & Automation* (2004) 3036-3042
- [8] “Artificial locomotion control: from human to robots” C. Azevedo, P. Poignet, B. Espiau. *Robotics and Autonomous Systems* 47 (2004) 203-223
- [9] “Assessing sensory function in locomotor systems using neuro-mechanical simulations” K. Pearson, Ö. Ekeberg, A. Büschges. *TRENDS in Neurosciences*, Vol 29 Issue 11 (2006) 625-631
- [10] “Ballistic walking” S. Mochon, T. Mahon. *Journal of Biomechanics*, Vol 13 (1980) 49-57
- [11] “Computational evolution of human bipedal walking by a neuro-musculo-skeletal model” K. Hase, N. Yamazaki. *Artif. Life Robotics* 3 (1999) 133-138
- [12] “Development of a human neuro-musculo-skeletal model for investigation of spinal cord injury” C. Paul, M. Bellotti, S. Jezernik, A. Curt. *Biol. Cybern.* 93 (2005) 153-170
- [13] “Energy efficient and robust rhythmic limb movement by central pattern generators” B. W. Verdaasdonk, H. F. J. M. Koopman, F. C. T. Van Der Helm. *Neural Networks* 19 (2006) 388-400
- [14] “Engineering entrainment and adaptation in limit cycle systems. From biological



inspiration to applications in robotics” J. Buchli, L. Righetti, A. J. Ijspeert. *Biol. Cybern.* 95 (2006) 645-664

[15] “Evolutionary generation of human-like bipedal locomotion”. K. Miyashita, S. Ok, K. Hase. *Mechatronics* 13 (2003) 791-807

[16] “Experimental study of biped dynamic walking” S. Kajita, K. Tani. *IEEE International Conference on Robotics & Automation* (1995) 13-19

[17] “Gait characteristics of a speed variable biped walking robot” H. Minakata, C. Katagiri, S. Tadakuma. *International Workshop on Advanced Motion Control, AMC* (2000) 542-547

[18] “Human balance and posture control during standing and walking” D.A. Winter PhD, *Peng Gait and Posture*: 1995; Vol. 3: pp 193-214, December

[19] “Neurobiological bases of rhythmic motor acts in vertebrates” S. Grillner. *Science* 228 (1985) 143-149

[20] “Modeling, stability and control of biped robots-a general framework” Y. Hurmuzlu, F. Génot, B. Brogliato. *Automatica* 40 (2004) 1647-1664

[21] “Normal Human Locomotion, Part 1: Basic Concepts and Terminology” Ed Ayyappa, MS, CPO. *Jour. of Prosthetics and Orthotic Prostheses* Vol. 9, Num. 1 (1997) 10-17

[22] “Predictive modelling of human walking over a complete gait cycle” L. Ren, R. K. Jones, D. Howard. *Journal of Biomechanics* (2006)

[23] “Reinforcement learning for quasi-passive dynamic walking of an unstable biped robot” K. Hitomi, T. Shibata, Y. Nakamura, S. Ishii. *Robotics and Autonomous Systems* 54 (2006) 982-988

[24] “Robustness of the dynamic walk of a biped robot subjected to disturbing external forces by using CMAC neural networks” C. Sabourin, O. Bruneau. *Robotics and Autonomous Systems* 51 (2005) 81-99

[25] “Synthesis of two-dimensional human walking: a test of the lambda-model” M. Günther, H. Ruder. *Biol. Cybern.* 89 (2003) 89-106

[26] “The anthropomorphic biped robot BIP2000” B. Espiau, P. Sardain. *Proceedings of the 2000 IEEE International Conference on Robotics & Automation* (2000) 3996-4001

[27] “Theories of bipedal walking: an odyssey” C. L. Vaughan. *Journal of Biomechanics* 36 (2003) 513-523

[28] “Understanding muscle coordination of the human leg with dynamical simulations” F. E. Zajac. *Journal of Biomechanics* 35 (2002) 1011-1018

## Annex A: How to create a stick figure

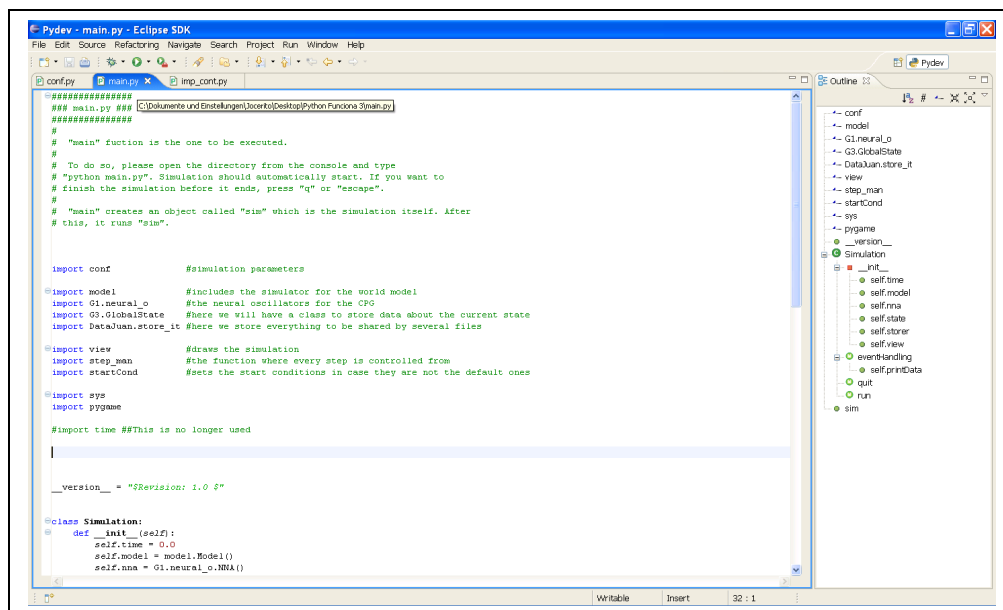
In this section we will describe the way stick figures can be generated. The process includes the use of several different programs and, so, a small guide of the process turns out to be necessary.

**Step 1:** Check the code in Python, inserting the necessary changes for our simulation

An editor will be needed to do so. Some of the most commonly used editors compatible with Python include:

- Eclipse<sup>2</sup>: an open source development platform. In order to work with Python code with this program you will need:
  - Java Runtime Environment (JRE)
  - Eclipse Integrated Development Environment (IDE)
  - a plug-in to use Python in Eclipse IDE, like PyDev
- GNU Emacs<sup>3</sup>
- Kate<sup>4</sup>: only available for Unix-like operating systems.

A complete list of development environments for Python can be found in:  
<http://wiki.python.org/moin/IntegratedDevelopmentEnvironments>



```
#####
### main.py ###
#####
#
# "main" function is the one to be executed.
#
# To do so, please open the directory from the console and type
# "python main.py". Simulation should automatically start. If you want to
# finish the simulation before it ends, press "q" or "escape".
#
# "main" creates an object called "sim" which is the simulation itself. After
# this, it runs "sim".

import conf          #simulation parameters
import model         #includes the simulator for the world model
import G1.neural_o   #the neural oscillators for the CGO
import G3.GlobalState #here we will have a class to store data about the current state
import DataAun.store_it #here we store everything to be shared by several files

import view          #draws the simulation
import step_man     #the function where every step is controlled from
import startCond    #sets the start conditions in case they are not the default ones

import sys
import pygame

#import time ##This is no longer used

|

__version__ = "$Revision: 1.0 $"

class Simulation:
    def __init__(self):
        self.time = 0.0
        self.model = model.Model()
        self.nna = G1.neural_o.NNA()
```

At this point, the variable “**drawDummy**” in `dummy.py` (in `Dummy` folder) must be set to “**False**”.

2 <http://www.eclipse.org/> 2008-02-28  
3 <http://www.gnu.org/software/emacs/emacs.html> 2008-03-03  
4 <http://kate-editor.org/> 2008-03-03

```
#####
### dummy.py ###
#####
#
# "dummy" includes all you need to get data from the program into a file so,
# after that, you can print it in Matlab.
#
# To do so, set drawDummy = True in conf.py, make sure nothing else is printed,
# execute "python main.py > data.data" and, once in Matlab, execute "load data.data"
# and "plotdummy(data)".
#
#
#
#
from math import pi, atan2, floor
drawDummy = False #True if you want to make a figure of the walking human
ScaleDummy = 0.00013
dummyTempo = 0.09 #Time elapsed between measurements of the dummy (in seconds)
dummySteps = 100 #Number of sets of data for drawing the dummy
dummyStepsI = 0

if drawDummy == True:
    dummyData = [0 for i in range(22)]

def printDummy(bodyData, theta):

    HIPx = bodyData.HIP.linx.getPosition()[0]
    HIPz = bodyData.HIP.linx.getPosition()[2]
```

**Step 2:** Make sure nothing is printed when the program is executed

To do so, execute the program in a command line interface.

```
Eingabeaufforderung
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Dokumente und Einstellungen\Jocerito>inipython.bat
C:\Dokumente und Einstellungen\Jocerito>set path=C:\Programme\MiKTeX 2.5\miktex\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\system32\wbem;%QUARTUS_ROOTDIR%\bin;%QESSE_ROOTDIR%\bin;C:\Programme\QuickTime\QTSystem\;c:\matlab6p5\bin\win32;c:\python24
C:\Dokumente und Einstellungen\Jocerito>cd Desktop
C:\Dokumente und Einstellungen\Jocerito\Desktop>cd "Python Funciona 3"
C:\Dokumente und Einstellungen\Jocerito\Desktop\Python Funciona 3>python main.py
C:\Dokumente und Einstellungen\Jocerito\Desktop\Python Funciona 3>_
```

**Step 3:** Activate plotDummy

Set **drawDummy** to “**True**”.

```
conf.py | main.py | imp_conf.py | dummy.py x
#####
### dummy.py ###
#####
#
# "dummy" includes all you need to get data from the program into a file so,
# after that, you can print it in Matlab.
#
# To do so, set drawDummy = True in conf.py, make sure nothing else is printed,
# execute "python main.py > data.data" and, once in Matlab, execute "load data.data"
# and "plotdummy(data)".
#
#
#
#
from math import pi, atan2, floor
drawDummy = True #True if you want to make a figure of the walking human
StartDummy = 0/0.00013
dummyTempo = 0.09 #Time elapsed between measurements of the dummy (in seconds)
dummySteps = 100 #Number of sets of data for drawing the dummy
dummyStepsI = 0

if drawDummy == True:
    dummyData = [0 for i in range(22)]

def printDummy(bodyData, theta):

    HIPx = bodyData.HIP.linx.getPosition()[0]
    HIPz = bodyData.HIP.linx.getPosition()[2]
```

**Step 4:** Print the data into a file

Go back to the command line interface and store the output data at the execution of the program in a file, as an array of numbers. In our example we have used the name data.data for this file.

```
Eingabeaufforderung - python main.py -
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Dokumente und Einstellungen\Jocerito>inipython.bat

C:\Dokumente und Einstellungen\Jocerito>set path=C:\Programme\MikTeX 2.5\miktex\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\system32\wbem;%QUARTUS_ROOTDIR%\bin;%QES_ROOTDIR%\bin;C:\Programme\QuickTime\QTSystem;c:\matlab6p5\bin\win32;c:\python24

C:\Dokumente und Einstellungen\Jocerito>cd Desktop

C:\Dokumente und Einstellungen\Jocerito\Desktop>cd "Python Funciona 3"

C:\Dokumente und Einstellungen\Jocerito\Desktop\Python Funciona 3>python main.py
-> data.data
```

**Step 5:** Load the data set in MATLAB so it can later be used.

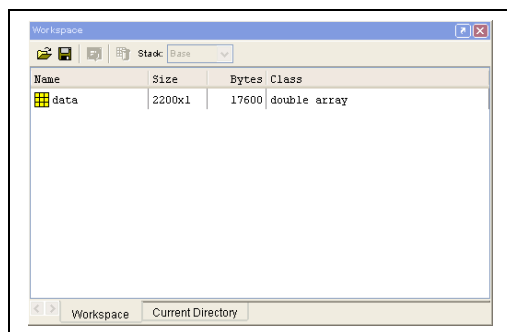
To do so, use “load” command at MATLAB console.

```
Command Window
Using Toolbox Path Cache. Type "help toolbox_path_cache" for more info.

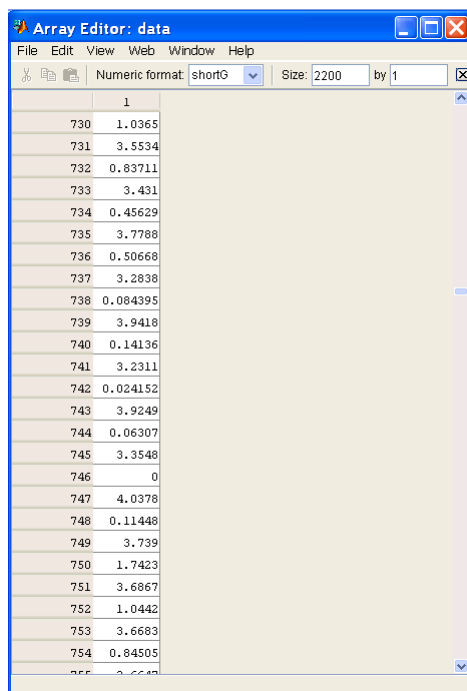
To get started, select "MATLAB Help" from the Help menu.

>> load data.data
>>
```

A new entry will appear in the **workspace**.

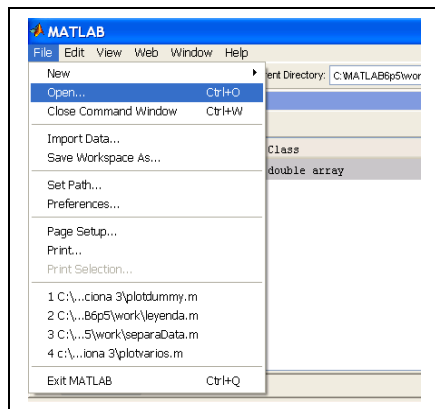


If you double-click on its name, an **Array Editor** window will appear, showing the array of data contained in it.

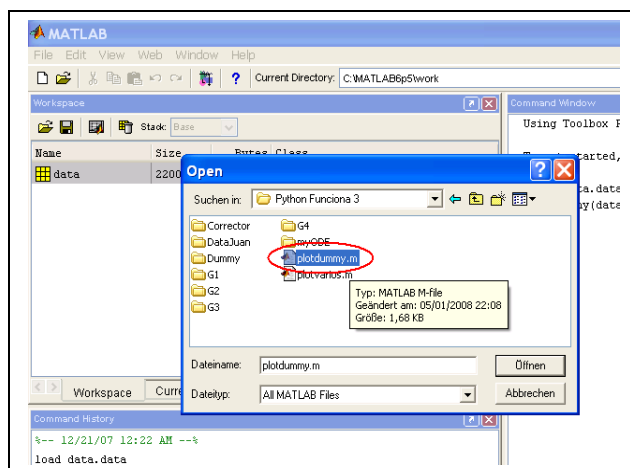


## Step 6: Open plotdummy.m

In the menu bar, click **File**, then **Open...**, or press **Ctrl+O**.



When the **Open** window appears, go to the folder where the file is and double-click it.



A text editor window will appear, showing the code of plotdummy.m.

```
c:\dokumente und einstellungen\jocerito\desktop\python funciona 3\plotdummy.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
1 function [ ] = plotdummy(data) Run
2 %PLOTDUMMY plots the walking pattern of the simulated
3 %
4 % To generate the data you must:
5 %
6 % 1.- set "drawDummy = True" in dummy.py
7 %
8 % 2.- execute "python main.py > data.data" in the console
9 %
10 % 3.- execute "load data.data" and "plotdummy(data)" in Matlab
11 %
12 %
13 %
14 % First, let's define some values:
15 [a, b] = size(data);
16 dummySteps = floor(a/22);
17
18
19 % Second, let's prepare the window for the drawing:
20 scrSz = get(0,'ScreenSize');
```

### Step 7: Configure plotDummy.m

First, configure the range of each axis that will appear in the figure.

```
43 plot([aux(13)+x, aux(17)+x],[aux(14), aux(18)], 'k') %LF0.segment1
44 plot([aux(11)+x, aux(19)+x],[aux(12), aux(20)], 'k') %RF0.segment3
45 plot([aux(13)+x, aux(21)+x],[aux(14), aux(22)], 'k') %LF0.segment3
46
47 scale = axis;
48 scale(1) = -0.2;
49 scale(2) = +12.2;
50 scale(3) = -0.2;
51 scale(4) = +3.5;
52 axis('square');
53
54 end
55
56
```

Then, configure the size of the plot window. It is recommended to fix the values so both axis have the same scale. This way, the results will be easier to analyze and won't lead to errors in interpretation.

```
18
19 % Second, let's prepare the window for the drawing:
20 scrSz = get(0,'ScreenSize');
21 %set(0,'DefaultFigureProperty',PropertyValue...)
22 set(0,'DefaultFigureName', 'Dummy Simulation')
23 set(0,'DefaultFigureNumberTitle','off')
24 figure('Position', [1 400 650 210])
25 plot([-10, 100], [0, 0], 'k')
26 hold on
27
28 % Now we can start drawing
29 aux = zeros(1,22);
30 cms = 0.0; %translation in y coordinates in order to see
```

## Step 8: Run plotdummy.m

This function must be run from MATLAB **console**. It produces a plot window containing the stick figure of the simulation. To do this, type **plotdummy(X)** in the **console**, where X is the name of the data file, as it appears in the **workspace**.

