Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2014

José Lucas Lemos Mendonça

Comportamentos para robôs humanóides simulados

Behaviours for simulated humanoid robots

Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

José Lucas Lemos Mendonça

Comportamentos para robôs humanóides simulados

Behaviours for simulated humanoid robots

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Nuno Panelas Nunes Lau, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Artur José Carneiro Pereira, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Luís Filipe de Seabra Lopes
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Luis Paulo Gonçalves dos Reis
Professor Associado da Escola de Engenharia da Universidade do Minho

Prof. Doutor José Nuno Panelas Nunes Lau
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos /
acknowledgements

Agradeço o tempo, disponibilidade e paciência aos que se cruzaram comigo no desenvolvimento da tese, desde os orientadores Nuno Lau e Artur Pereira, à equipa do FC Portugal 3D, Rui Ferreira, Abbas, Nima e aos meus colegas.

Palavras Chave

Humanóide, Bípede, Optimização, Ferramentas de desenvolvimento e depuração, Simulação de Futebol, Robô

Resumo

Esta tese está inserida na equipa FC Portugal 3D, que compete na liga de futebol robótico simulado 3D. Os objetivos da tese são melhorar os comportamentos já existentes e desenvolver ferramentas de suporte ao desenvolvimento e depuração para o agente robótico.

Nesse sentido, foi melhorado o processo de optimização de comportamentos de forma a torná-lo mais eficiente e adaptado para incluir os novos modelos heterogéneos disponibilizados. Ao executar o processo de optimização, usando o algoritmo de estado de arte CMA-ES, foi obtido reduções para metade do tempo nos comportamentos de levantar-se. Seguidamente o agente foi colocado a correr em modo síncrono, o que permite que as simulações corram à velocidade de processamento do computador em uso, e não à velocidade da simulação da competição em que cada ciclo demora 20ms. Assim é possível executar simulações e consequentemente inferir conclusões muito mais rapidamente.

Passou-se a usar a informação de giroscópio e o cálculo dos ângulos de euler para obter uma melhor estimativa da rotação do robô. Por outro lado, devido ao lançamento de novos tipos de robôs, a arquitectura do agente teve de ser atualizada e novos comportamentos foram criados e optimizados para estes novos modelos. Em relação ao modelo original, alguns comportamentos são executados mais rapidamente e melhor pelos modelos novos, devido às suas alterações físicas. Por fim, nos comportamentos foi dada a possibilidade de definir pré condições em etapa do mesmo, para que possa ser abortado caso as condições não se verifiquem. Esta alteração veio reduzir o tempo desperdiçado a executar a totalidade do comportamento em situações em que não é provável o seu sucesso .

Em termos de ferramentas, foi colocada uma Janela de Monitor de Agente para cada agente que, apresenta em tempo de simulação variáveis que o código do agente disponibiliza, interage com código através de widgets de seleção ou preenchimento, e se a simulação estiver a correr em modo síncrono, permite definir o tempo de ciclo da simulação, pausá-la e executar ciclo a ciclo, o que permite vantagens óbvias em termos de análise de execução dos agentes. Seguidamente, foi criada uma ferramenta de teste para comportamentos definidos em XML, que permite, em tempo de execução, alterar o ficheiro a testar, alterar o seu conteúdo, agrupar vários ficheiros em sequências e executar vários agentes em paralelo. Por fim, a última ferramenta é um Analizador de Logs gerados pelos agentes e pelo simulador que permite, entre outras funcionalidades, ver em forma de gráficos variáveis da simulação, exportar para diferentes formatos, filtrar a simulação usando informação da mesma e correr um servidor de forma a ser possível analizar em paralelo, gráficos de variáveis escolhidas e a simulação num visualizador.

Keywords

Humanoid, Biped, Optimization, Development and Debugging Tools, Soccer Simulation, Robot

Abstract

This thesis in inserted in the FC Portugal 3D team, which competes in the humanoid simulation league 3D from RoboCup. The objectives of this thesis are to improve the behaviours already created and to develop tools to support the development and debugging of the robotic agent.
With this in mind, the process of optimization was improved to make it more efficient and adapted to include the new heterogeneous models. Executing the optimization process, using the state of the art algorithm CMA-ES, the time of the getup was reduced by half. Afterwards, the agent was put running in sync mode, which allows the simulations to run as fast as the computer in use can process, and not the simulation speed of the competion with cycles of 20ms.
In the agent posture, it is now used the information from the gyroscope and the euler angles are calculated to get a better estimative of the robot orientation.
On the other hand, the agent architecture was updated and new behaviours were created and optimized to support the new heterogeneous models. In relation to the standard model, some behaviours execute faster because of their physical difference.
In the slot behaviours, it is now possible to defined preconditions in each step, so the agent can abort the behaviour when any condition does not comply. This change reduces the time wasted executing all the behaviour in situations in which the success is improbable.
In terms of tools, a Agent Monitor Window was created for each agent which can: present in runtime variables from the agent code; interact with the code trough widgets; and if the simulation is in sync mode, defined the simulation cycle time, with the possibility to pause it and execute step by step, which gives a great advantage in terms of analysing the agent execution. The second tool was a behaviour testes for behaviours defined in XML, which allows, in runtime, to change the behaviour to test, edit its content, aggregate different files in sequence and finally the tolls can execute various agents in parallel.
The last tools is Log Analyser of the logs generated by the agents and the server, which allows: exporting in different formats, see in form of plots the variables parsed, filtrate the simulation information; and create a server simulation which can be used to analyse, in parallel, the plots of chosen variables and the simulation in a monitor.

Contents

List of Figures

List of Tables

List of Acronyms

PID
Proportional Integral Derivative

TCP
Transmission Control Protocol

CPG
Central Pattern Generator

CoP
Center of Pressure

CM
Center of Mass

GCoM
Ground projection of the Center of Mass

ZMP
Zero Moment Point

FZMP
Fictitious Zero Moment Point

PSO
Particle Swarm Optimization

CMA-ES
Covariance Matrix Adaptation - Evolution Strategy

SPL
Standard Platform League

SMN
Static Mesh Node

AI
Artificial Intelligence

RoboCup
Robot Soccer World Cup

XML
Extensible Markup Language

RDS
Ruby Diff Scene

RSG
Ruby Scene Graph

GTK
GIMP Toolkit

GUI
Graphical User Interface

GRAL
GRAphing Library

WBMP
Wireless Application Protocol Bitmap Format

BMP
Bitmap image file

PNG
Portable Network Graphics

JPEG
Joint Photographic Experts Group

PDF
Portable Document Format

GIF
Graphics Interchange Format

SVG
Scalable Vector Graphics

EPS
Encapsulated PostScript

Simspark
Spark Generic Physical Multiagent Simulator

chapter 1

Introduction

1.1 Motivation

In the Robot Soccer World Cup (RoboCup) initiative there are researchers trying to push the area of Artificial Intelligence and Robotics further. The environment of robotic soccer is a great benchmark to test the progress made in the area and each league has its set of problems and research focus. In this thesis the focus is in the 3D Simulation League, in which it is necessary to create a soccer humanoid agent, with the challenges that come with it.

For the aforementioned initiative, the FC Portugal[1] team was created. Since 2000, it participated in 2D and 3D and had achieved some great results. Recently the team got 4th place in 2014 RoboCup, 1st in Robocup German Open and 1st in Robotica 2014 in Portugal.

Writing an agent that behaves autonomously and cooperates with its teammates is a difficult task, but the 2D simulation got some good results in that area. On the other hand, the 3D simulation, besides still having to deal with cooperation and coordination, has also does face the challenge of having to control humanoid models. These models have particular needs, as the bipedal locomotion is difficult to achieve. Furthermore, the biped humanoid agent not only needs to walk stably, it also needs to walk and rotate in all directions, kick and pass the ball, getup, and any other behaviour that is expected from a humanoid soccer player.

1.2 Objectives

In the context of the FC Portugal team and the 3D Simulation League, the objectives of this thesis are:

- Develop robust and efficient behaviours of kick, getup, dribble and any other behaviour needed, using manual control of the joints, or inverse kinematic or using optimization techniques.
- Create tools to support the development and debugging process, make it more efficient and/or easier.

1.3 Structure

The structure of this thesis is as follows:

- Chapter 1: As seen, this chapter presents this thesis motivation and its objectives
- Chapter 2: Present the RoboCup, its leagues, challenges and software used.
- Chapter 3: Description of the problems faced when working in humanoid behaviours, some solutions and concepts used nowadays, and some RoboCup teams research in that area.
- Chapter 4: Brief description of some optimization algorithms used in the FC Portugal team.
- Chapter 5: Improvements in the optimization process and its results.
- Chapter 6: Adaptation to the new heterogeneous models, use of the gyroscope and addition of conditions in some behaviours.
- Chapter 7: Tools developed to make development process easier and quicker.
- Chapter 8: Present the conclusion and some future work that can be done in sequence to this thesis.

chapter 2

RoboCup Simulation3D

This chapter outlines the RoboCup project and its Simulation 3D competition, where this thesis is inserted, and execution environment.

Founded in 1997, RoboCup is a project to promote Artificial Intelligence (AI) and robotics research. It has set a challenging long term goal:

"By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup." [2][3]

For that purpose, there is an annual international robotics competition an integrated research task covering broad areas of AI and robotics. Such areas include: real-time sensor fusion, reactive behaviour, strategy acquisition, learning, real-time planning, multiagent systems, context recognition,vision, strategic decision-making, motor control, intelligent robot control.

In order to reach the proposed goal, various competitions were created, separating the research tasks, each one of them focusing in a different set of problems.

- Middle-Size League
- Small-Size League
- Standard Platform League
- Humanoid League
- Simulation League

Besides soccer, there are others leagues with different lines of development in the form of rescue challenges, either simulated [4] of real [5], and challenges aimed to develop service and supportive robot technology with high relevance for future personal domestic applications [6].

In this competitions, the physical ones have some disadvantages, inherent of using physical models:

- The robot is subject to be damaged during its usage or even during its transporting, in addition to sometimes being costly to repair.
- The testing process is slow. Any change has to be loaded in the robot and its test runs at a slow rate.

- It is costly to always buy the newer versions, year after year, or just new components for repairing.
- Multiple team members cannot work at the same time on one robot.

To address these issues, a simulated environment is a good option, but it needs to be as close as possible to the physical environment. The RoboCup 3D Simulation Soccer League is a competition where software agents control humanoid robots in a soccer game. The platform tries to simulate the rules and physics of a soccer game, reproducing physical robot limitations, like hinges movement restriction, sensors noise,etc. The soccer field dimensions are 30 by 20 meters as we can see in image 2.1.

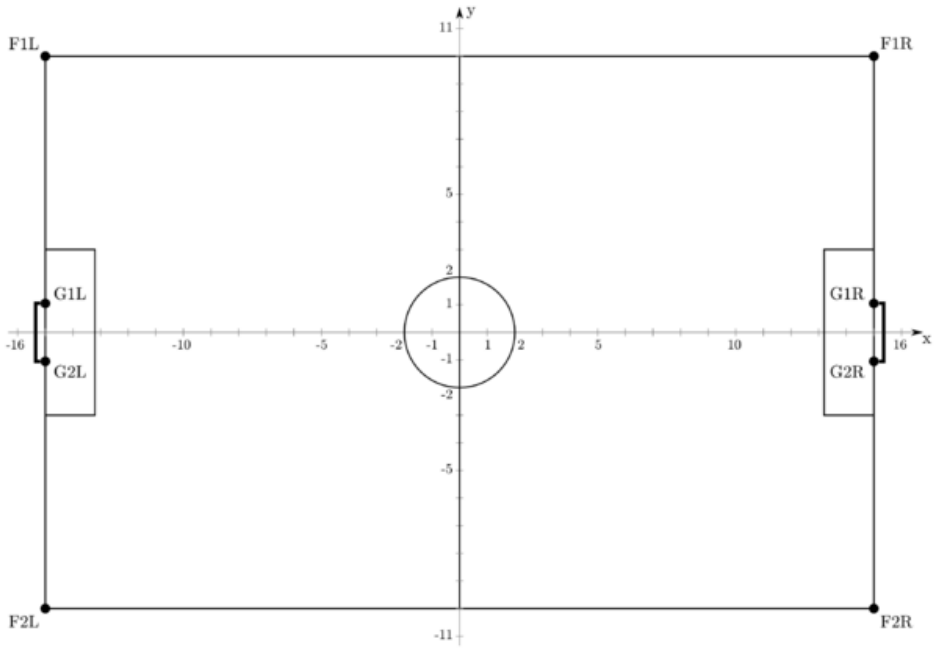


Figure 2.1: Field dimensions.

In 2004, the 3D Simulation was introduced, beginning with a spherical agent model. The first humanoid model was the Fujitsu HOAP-2, which changed the development from strategic behaviours to low level control and basic behaviours like walk, kick, getting up, turn, etc.

In 2008, both the Standard Platform League (SPL) and the 3D Simulation began to use the NAO robot [7] from Aldebaran as their model. This permitted the researchers to try their development in the simulated NAO version before putting the code in the physical robots from the SPL.

The number of robots in the game increased over the years, till 2012 when 11 vs 11 games where implemented. In 2013 the teams were able to use heterogeneous robot types, which are variations of the standard NAO robot so the development would not be attached to only one humanoid model. With the new models, some behaviours work better in some models than others, making way to a new strategic configurations.

Besides soccer simulation, other challenges were created to promote the resolution of some specific problems. In 2013 the first Drop In Player Challenge proposes a game where there is one agent per league team, so the communication between teammates is crucial. In 2014 the first running challenge occurred, pushing the teams to develop running behaviours to be faster and more human alike.

2.1 Simspark and Rcssserver3d

In order to run the Robocup3D simulation Spark Generic Physical Multiagent Simulator (Simspark) server was the elected tool. It is a generic physical multi-agent simulator system for agents in three-dimensional environments built on the flexible Spark application framework. On top of that, the rcssserver3d was created, which is a 3D soccer simulation with the rules set for the RoboCup 3D simulation league.

2.1.1 Monitor

While running the simulation, it is possible to visualize it by using a Simspark monitor. It connects to a running Simspark server, then starts to receive periodic messages describing the simulation state. Besides the game visualization, the messages contain the score, current playmode, time, etc. Furthermore, the monitors can send commands to the server stated in the next section or play back recorded Log Files. There are three forms to use monitors:

- If network overhead is not intended, one can configure the Simspark server to render the simulation itself.
- rcssmonitor3d is the basic monitor that comes with rcssserver3d. It can render the game and show the game info, but it is somewhat limited.
- Another monitor used by the community is RoboViz [8] [9]. It has enhanced visualization capabilities and it was the one used in this thesis. Some of its features are:
  - Enhanced Graphics which give a new dimension to the simulation and its move attractive to the spectators.
  - Interaction and control of the agents, ball position, switching play modes.
  - Debugging visual elements can be displayed such as circles, text and lines, which help in the development process.

We can see the graphics difference between the two monitors in figure 2.2.

Figure 2.2: RoboViz(Left) vs rcssmonitor3d(Right)

2.2 Network Protocol

The Simspark communications are based in Transmission Control Protocol (TCP) connections between nodes, where messages in the form of S-expressions [10] are exchanged. These expressions are well known in Lisp programming language for coding and data declaration, being easy to parse and are readable by humans.

In each message there is a 32 bit unsigned integer in network order, where the length of the payload is declared. Furthermore, the messages use the default ASCII character set, which means one character is encoded in one byte. [11]

The network communications are divided in two types, presented in the following sections.

2.2.1 Server/Agent

In this type of communication the server sends messages to the agent containing the agent's perceptors, hinge positions, heard messages, seen objects. In response, the agent sends message with commands to be applied to its effectors and its beam.

2.2.2 Server/Monitor

The Simspark server binds to TCP port 3200 by default, where it listens for monitors connections. When a monitor connects to it, the information arrives at the monitor in the following order:

- An environment information message followed by the full scene graph. A example message is as follows

```
(
(FieldLength 18)
(FieldWidth 12)
(FieldHeight 40)
(GoalWidth 2.1)
(GoalDepth 0.6)
(GoalHeight 0.8)
(FreeKickDistance 1.3)
(WaitBeforeKickOff 2)
(AgentRadius 0.4)
(BallRadius 0.042)
(BallMass 0.026)
(RuleGoalPauseTime 3)
(RuleKickInPauseTime 1)(RuleHalfTime 300)
(play_modes BeforeKickOff KickOff_Left KickOff_Right PlayOn
KickIn_Left KickIn_Right corner_kick_left corner_kick_right
goal_kick_left goal_kick_right offside_left offside_right
GameOver Goal_Left Goal_Right free_kick_left free_kick_right)
)
```

Figure 2.3: Environment information message example

- A game state message followed by the full scene graph [12].

- Periodically sends partial game state messages followed by a full or partial scene graph, depending on what has been updated lately. The rate at which the server sends messages is defined in the file `spark.rb` which is located in the installation directory.

For trainer purposes the monitors can send commands such as:

- Moving an Agent
- Positioning the Ball
- Setting the Play Mode
- Drop the Ball
- Kick Off
- Select Agent
- Kill Agent
- Repositioning an Agent
- Ack

The partial or full scene graph are composed by nodes of different types in order to describe the simulation scenes. Two important nodes types used further in the thesis for developing a debugging tool are the Transform and Geometry ones:

- **Transform node**
  This node represents a 4x4 homogeneous transformation matrix 2.5 used to represent a geometric transformation. It defines a translation, a rotation and a scaling factor. To represent a reference frame in three dimensional space (see figure 2.4), a homogeneous transformation matrix (see figure 2.5) has three mutually perpendicular unit vectors: `n,o,a`, which stand for normal, orientation and approach, respectively .

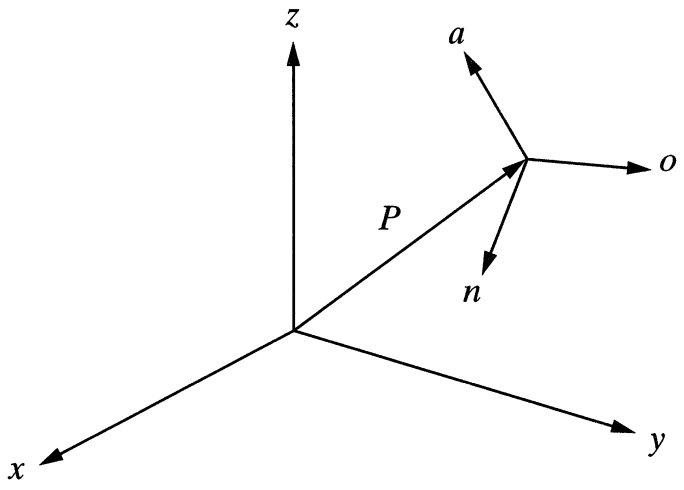


Figure 2.4: Representation of a Frame in a Fixed Reference Frame [13]

```
[nx ox ax Px]
[ny oy ay Py]
[nz oz az Pz]
[ 0  0  0  1]
```

Figure 2.5: Homogeneous transformation matrix

In Simspark, this transformation matrix is represented as:
(nd TRF (SLT nx ny nz 0 ox oy oz 0 ax ay az 0 Px Py Pz 1)) where *TRF* stands for Transform and *SLT* for Set Local Tranform which is a ruby function for setting the local transformation of a given node in the scene graph.

- **Geometry node**
  These nodes are used to describe the objects models, materials and scales. They are divided in two types:
  - **StaticMesh**
    Defines a mesh which should be loaded from a *.obj* file located in the Simspark path. As a example we can see2.6 and 2.7.

```
(nd StaticMesh
  (load <model>)
  (sSc <x> <y> <z>)
  (setVisible 1)
  (setTransparent)
  (resetMaterials <material-list>)
)
```

Figure 2.6: Static Mesh node format

```
(nd StaticMesh
  (load models/naohead.obj)
  (sSc 0.1 0.1 0.1)
  (resetMaterials matLeft naoblack naogreynaowhite)
)
```

Figure 2.7: Static Mesh node example

  - **SMN**
    Defines a mesh using one of the predefined models of the Simspark: *StdUnitBox*, *StdUnitCylinder*, *StdUnitSphere*, *StdCapsule*. A example is present in figure 2.8 and 2.9.

```
(nd SMN
  (load <type> <params>)
  (sSc <x> <y> <z>)
  (setVisible 1)
  (setTransparent)
  (sMat <material-name>)
)
```

Figure 2.8: SMN format

```
(nd SMN
  (load StdUnitCylinder 0.015 0.08)
  (sSc 1 1 1)
  (sMat matDarkGrey)
)
```

Figure 2.9: SMN node example

2.3 Summary

Ending this chapter, the reader should be familiar with the simulation environment components and its network protocol. A brief description of the messages exchanged between server-agents and server-monitors is presented to help understand the type of interaction between components.

chapter 3

Humanoid Behaviours

This chapter presents some concepts used in humanoid behaviours an some FC Portugal 3D current work on behaviours.

Humanoid models present multiple complex challenges such as the creation of stable behaviours in different circumstances. In order to create an omnidirectional walk, a kick or a getup behaviour there are various methods and approaches to consider.

3.1 Stability

For creating a stable and robust behaviour there needs to be some form of control, so the robot mantains a motion in which it will not fall. The most used stability criteria are the Center of Mass (CM), Center of Pressure (CoP) and Zero Moment Point (ZMP)

3.1.1 Center of Mass

The CM is the point where all of the mass of the object is concentrated. It represents the mean position of the matter in a body. Normally, it is in the CM that external forces are considered to be applied.

In a system of particles the CM is calculated as follows:

$$R = \frac{1}{M} \sum_{i=1}^{n} m_i r_i \tag{3.1}$$

where

- R denotes the coordinates of the center of mass
- n denotes the number of particles
- m_i denotes the mass of the particle i
- r_i denotes the coordinates of the particle i

- M is the sum of the masses of all the particles

In a volume V with a continuous mass distribution:

$$R = \frac{1}{M} \int_V \rho(r) r dV \tag{3.2}$$

where

- R denotes the coordinates of the center of mass
- $\rho(r)$ denotes the density of a mass within the volume V
- r denotes the coordinates of the mass within the volume V
- M denotes the total mass of the volume

The projection of the CM in the ground it is known as Ground projection of the Center of Mass (GCoM). This criterion it used, for example, in static gait to check stability. For that, the GCoM must be in the foot-support area.

3.1.2 Center of Pressure

The CoP in the humanoid model is the point where the sum of all the forces between its feet and the ground are applied. The forces may be obtained from the force-torque sensors at the feet of the robot. In addition to the CM, normally the CoP its used to measure balance in bodies.

3.1.3 Zero Moment Point

The ZMP represents the point in the ground where the total of horizontal inertia and gravity forces equals zero. In other words p is the point where $T_x = 0$ and $T_y = 0$, where T_x, T_y represent the moments around x- and y-axis generated by reaction force F_r and reaction torque T_r , respectively. When ZMP exists within the domain of the support polygon (see figure 3.1), the contact between the ground and the support leg is stable [14].

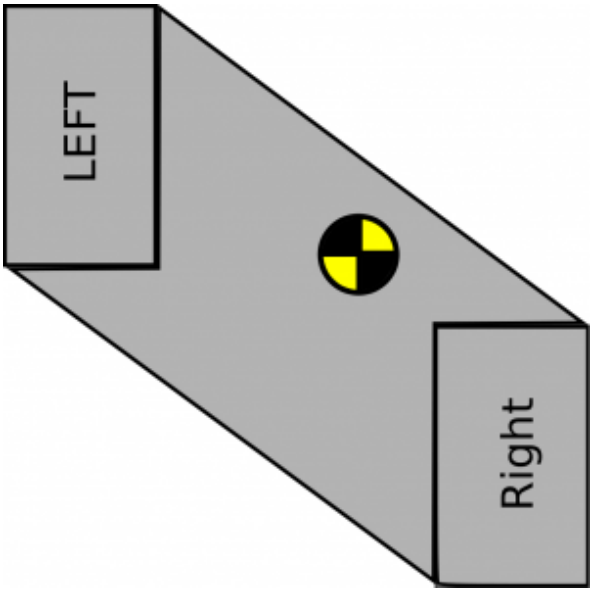


Figure 3.1: ZMP support polygon

3.1.4 Static vs Dynamic Stability

Using static stability the GCoM is maintained inside the support polygon so the robot does not fall [15] [16] [17]. The support polygon is the convex hull of the foot support area. For this, the robot must adjust its posture very slowly to minimize the dynamic effects [18]. The support polygon varies during the walk; it is the contact area between the foot and the ground when the robot has only one foot on the ground (single-support phase) and the convex hull of both contact areas between the feet and the ground, when both feet are on the ground (double-support phase).

In the human walk there is no static stability, since the humans walk in a state of constant falling, falling forward and catching themselves using the swinging foot while continuing to walk forward. In this falling movement, the GCoM moves forward getting outside the support polygon, without expending energy to adjust itself. However, it has dynamic stability because it results in a stable walk if the walking motion is continuous. Its stability is assured by maintaining the ZMP or CoP inside the support polygon. In a dynamically stable walk, the ZMP coincides with the CoP [14] [19]. When the ZMP leaves the support polygon the gait is not dynamically stable because the ground cannot exert the forces needed to keep it from rotating around one of the edges of the support polygon. The ZMP outside the support polygon is called the Fictitious Zero Moment Point (FZMP) and its distance to the foot edge is proportional to the intensity of the instability.

The advantages of static stability is its simplicity and that the robot can pause its motion at any moment of the gait stably. However, it makes the walk slow and generally leads to more power consumption since the robot has to adjust its posture so that the GCoM is always inside the support polygon. On the other hand, using dynamic stability generally leads to faster and reliable walking gaits.

3.1.5 Other Approaches

The HFutEngine3D team use a 3D linear inverted pendulum to plan the walking pattern and L3SIM, Mithras3D the ZMP and CM to detect stability in motion [20] [21][22]

The RoboCanes team uses genetic algorithms to optimize motions created with sequence of keyframes containing joint angles, and uses CMA-ES and PSO for making ***kinect*** based motions robust [23]

3.2 FC Portugal

During the years of research in the FC Portugal, different approaches were implemented and tested to tackle the challenges of walking, kicking the ball, getup, etc. The following sections present the most recent ones.

3.2.1 Slot Behaviour

This type of behaviour was implemented by Hugo Picado [24] as a simple version of the method proposed in [25]. Basically it define an interpolation using sin functions over an amount of time between the current and the target angles.

The definition of the behaviour is made using slots. They correspond to an interval of time from 0 to δ where multiple joints can be moved in parallel. It's possible to define multiple slots sequentially, each one with its δ interval of time. In each, besides the initial and final angle, one can define the initial and final angular velocities, and the parameters of the Proportional Integral Derivative (PID). So, for each joint, the trajectory is generated as follows:

$$f(t) = A * \sin\left(\frac{\phi_f - \phi_i}{\delta}t + \phi_i\right) + \alpha, \forall t \in [0, \delta] \tag{3.3}$$

where

- $f(t)$ is the trajectory function
- δ is the duration of the slot in milliseconds
- ϕ_i is the initial phase (influence the initial angular velocity)
- ϕ_f is the final phase (influence the final angular velocity)
- A is the amplitude
- α is the offset

A and α are calculated as follows:

$$A = \frac{\theta_f - \theta_i}{\sin(\phi_f) - \sin(\phi_i)} \tag{3.4}$$

$$\alpha = \theta_i - A * \sin(\phi_i) \tag{3.5}$$

where θ_i and θ_f are respectively, the initial and final angles, which should be defined between $-\pi$ and π. A slot behaviour example is presented in figure 3.2.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE joints [
<!ENTITY head1  "0"     >
...
<!ENTITY rarm4  "21"    >
]>

<behavior name="FallBack" type="SlotBehavior">

  <slot name="ForceFalling" delta="0.5" >
    <move id="&head1;" angle="0"    />
    <move id="&head2;" angle="0"    />
    <move id="&lleg1;" angle="0"    />
    <move id="&rleg1;" angle="0"    />
    <move id="&lleg2;" angle="0"    />
    <move id="&rleg2;" angle="0"    />
    <move id="&lleg3;" angle="0"    />
    <move id="&rleg3;" angle="0"    />
    <move id="&lleg4;" angle="0"    />
    <move id="&rleg4;" angle="0"    />
    <move id="&lleg5;" angle="-45"    />
    <move id="&rleg5;" angle="-45"    />
    <move id="&lleg6;" angle="0"    />
    <move id="&rleg6;" angle="0"    />
    <move id="&larm1;" angle="-90"  />
    <move id="&rarm1;" angle="-90"  />
    <move id="&larm2;" angle="0"   />
    <move id="&rarm2;" angle="0"  />
    <move id="&larm3;" angle="0"  />
    <move id="&rarm3;" angle="0"   />
    <move id="&larm4;" angle="0"    />
    <move id="&rarm4;" angle="0"    />
  </slot>
  <slot name="ResetPositionAndWait" delta="0.5">
    <move id="&lleg5;" angle="0"    />
    <move id="&rleg5;" angle="0"    />
  </slot>

</behavior>
```

Figure 3.2: Slot behaviour example

3.2.2 Central Pattern Generators Behaviour

Central Pattern Generator (CPG) is a neural oscillator that produces rhythmic patterned outputs without the need for any rhythmic input [26] [27]. In biology, there are many animals that have CPG for their behaviours (i.e human walking), with different CPG controlling different limbs. The generator doesn't need sensory feedback information to generate its output, but it could be used to correct motion and/or do compensation [28].

In robotics, by defining a mathematical model it's possible to simulate these biological neural oscillators. Based on that, Sven Behnke created an omnidirectional walk [29] with walk direction, speed and rotational speed as input. Despite that, normally it is hard to determine the parameter configuration that generates the desired walking pattern.

A CPG example is presented in figure 3.3.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE joints [
<!ENTITY head1  "0" >
...
<!ENTITY rarm4  "21"    >
<!ENTITY amp1 "12">
<!ENTITY amp2 "24">
<!ENTITY amp3 "10">
<!ENTITY amp4 "4">
<!ENTITY amp5 "24">
<!ENTITY period "0.25">
]>

<behavior name="SideRight" type="CPGBehavior">

  <patterns>
    &lleg2;: -&amp3; &period; 1.570796327 0;
    &rleg2;: &amp3; &period; 1.570796327 0;

    &lleg3;: -&amp1; &period; 0 28.44;
    &rleg3;: &amp1; &period; 0 28.44;

    &lleg4;: &amp2; &period; 0 -46.33;
    &rleg4;: -&amp2; &period; 0 -46.33

    &lleg5;: -&amp1; &period; 0 31;
    &rleg5;: &amp1; &period; 0 31;

    &lleg6;: &amp3; &period; 1.570796327 0;
    &rleg6;: -&amp3; &period; 1.570796327 0;

    &lleg1;: &amp4; &period; 1.570796327 0;
    &rleg1;: &amp4; &period; 1.570796327 0;

    &larm1;: 0 &period; 0 -90;
    &rarm1;: 0 &period; 0 -90;
  </patterns>

  <delta>&period;</delta>

</behavior>
```

Figure 3.3: CPG example behaviour

3.2.3 Omnidirectional walk

From a turn and front walk to navigate the field, to CPG [24], passing by Truncated Fourier Series [30], the walk behaviour has seen some implementations. The current one is the Nima Shaffi omnidirectional walk based on the ZMP criterion for its stability [31]. In this approach, the biped walking trajectory is derived from the desired ZMP by computing the feasible CM trajectory .This trajectory is calculated using an approximation for the dynamics of the biped robot, the 3D linear inverted pendulum model [32].

So, the architecture is divided in different modules as follows 3.4:

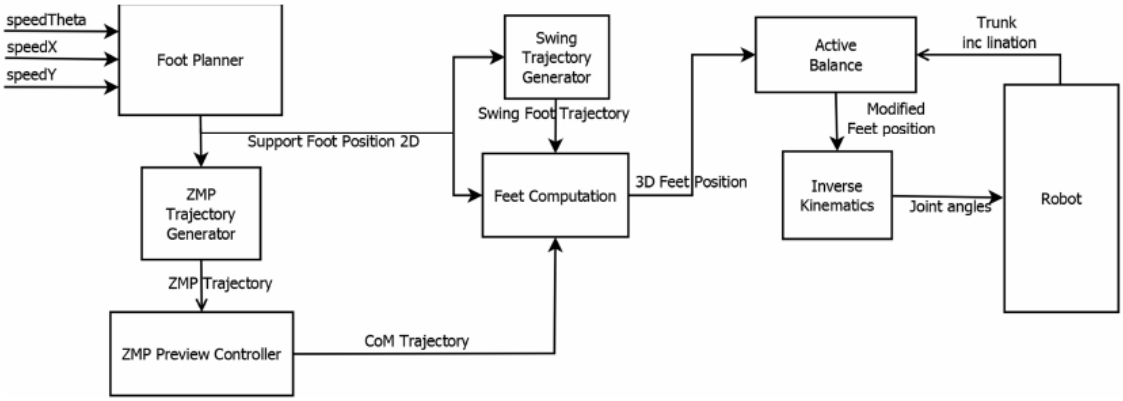
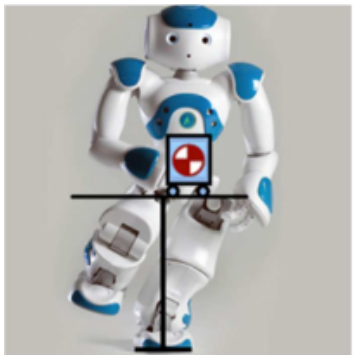
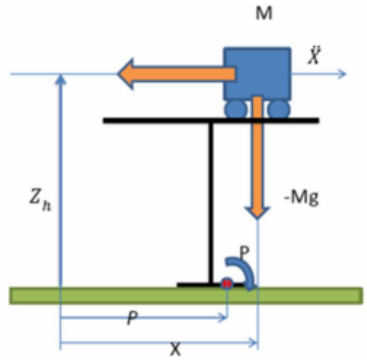


Figure 3.4: Omnidirectional walk architecture

The input of the walk is the desired speed in X,Y and its angle θ. Based on that input and the restrictions, like the foot reachability and feet inner collision, the foot planner generates the future support steps positions in 2D. Using the planned steps, the support polygon of ZMP and its position, the ZMP trajectory is calculated. From there, the Cart-table model (see figure **??**) is used to project the possible body swing and its CM trajectory.



(a) Cart-table in a humanoid robot



(b) Cart-table schematic

Figure 3.5: Omnidirectional walk Cart-table model from [31]

This model assumes that all masses are concentrated on the cart and the support legs doesn't have any mass. The simplification is not far from reality, as the legs have normally less mass than the upper body.

However, to apply the model, one must solve its differential equations, whose solution consists of unbounded hyperbolic cosine functions, and the CM trajectory is very sensitive to time step variation of the walk.

So, another CM trajectory generation possibility is using ZMP Preview Controller, based on the Kajita work [33] extended with the Park method [34]. Its equation and diagram 3.6 is:

$$u(k) = -G_i \sum_{i=0}^{k} e(i) - G_x x(k) \tag{3.6}$$

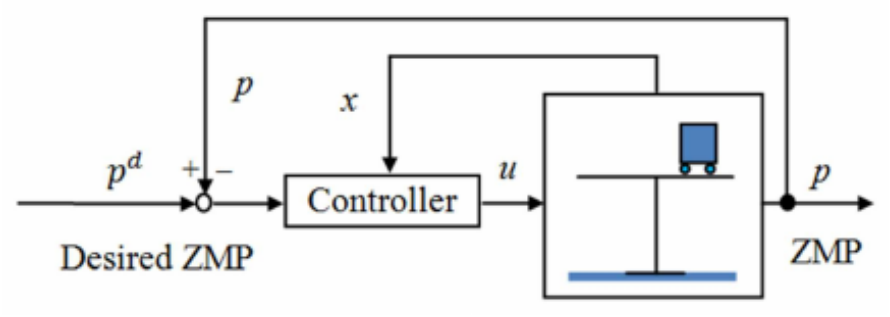


Figure 3.6: Preview controller

where

- G_i is the gain for the ZMP tracking error
- G_x is the gain for state feedback
- $e(i) = p - p^d$ is the controller error
- $x(k)$ is the position of the CM in the k sample time
- k denotes the k^{th} sample time
- p^d is the desired ZMP position
- p is the ZMP position calculated from the cart-table model

With this controller is not sufficient to follow the reference ZMP because of the phase delay. For that, preview samples of ZMP in the future are used as follows:

$$u(k) = -G_i \sum_{i=0}^{k} e(i) - G_x x(k) - \sum_{j=1}^{N_L} G_p p^d(k+j) \tag{3.7}$$

where

- N_L is the number of samples in the future used
- G_p is the preview gain
- $p^d(k+j)$ is the ZMP previewed $k+j$ in the future

Even after the walking trajectory calculations are done with the use of ZMP criteria, the stability is not totally guaranteed because of the leg's effectors and the simplification of the cart-table. So, before sending the final feet position to the Inverse Kinematics module, there's a need for an active balance module. It tries to maintain the trunk still with zero pitch and roll. For that, it makes use of a PID controller with the measures from the robot inertial measurement unit as input.

3.2.4 Omnidirectional Kick

Before 2012, the team was using a Slot Behaviour for its kick, which consisted in keyframes defining the motion required. This basically was a serie of static values for the joints and the movement was interpolated between keyframes. The problem with static values is that it makes the behaviour very

inflexible, working with a slow preparation phase, where the robot positions itself in a predefined position in relation to the ball (based on the desired direction) to kick it forward.

So, in 2012, Rui Ferreira tried a new approach to make the kick more flexible and to give control of the kick direction [35]. With this approach, the trajectory is created using Bézier curves (see figure 3.7 and table 3.1) [36], steering the kicking foot to the ball, so it will be kicked in the intended direction. This trajectory is updated in case of any ball movement within the foot range.

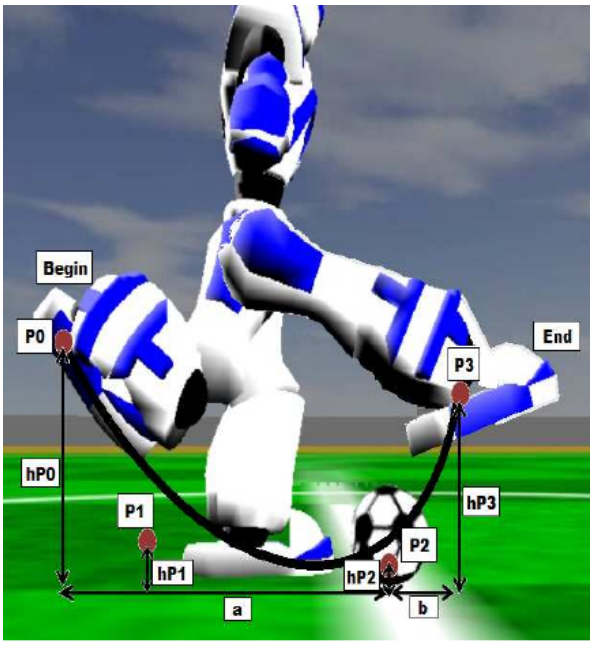


Figure 3.7: Kick parameters

Parameter	Description
a	Distance between the ball and the curve start
b	Distance between the ball and the curve end
hP0 hP1 hP2 hP3	Bézier cubic curves parameters
duration	Kick duration
Foot Orientation	Angle between foot orientation and vector Ball2Target, so it's possible to kick using different parts of the foot

Table 3.1: Parameters Description

With this parameters the trajectory is created. A global view of the architecture is presented in 3.8.

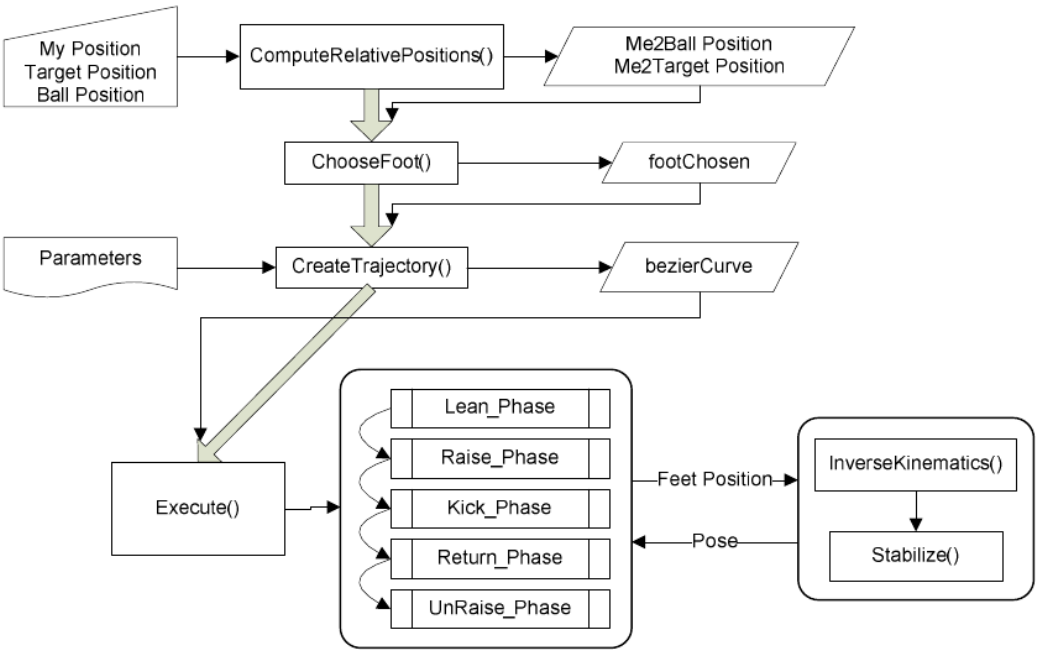


Figure 3.8: Omnidirectional kick sequence

This sequence unfolds into 5 phases. They are:

Lean_Phase

The robot shifts its CM to the support leg.

Raise_Phase

The robot raises the kick foot to the curve start.

Kick_Phase

The robot kicks the ball, executing the created trajectory.

Return_Phase

The kick leg returns to the base position, without touching the ground.

UnRaise_Phase

The robot shifts his CM to both legs, putting the kick foot in the ground.

Furthermore, there is the **Inverse Kinematics** module to calculate the leg joints values based on the execution output and the **Stability** module which tries to stabilize the robot while performing the movement.

3.3 Summary

Ending this chapter, the reader should be familiar with some concepts used in humanoid behaviours and the FC Portugal 3D behaviours work.

chapter 4

Population Based Optimization

In the elaboration of a behaviour there are multiple variables and constants involved. If manually defined, these values are not optimal, just ones that work more or less in each case. Thus, there is potential to change the values so the result of the behaviour has better performance. Instead of manually trying each set of values, an optimization procedure can be applied in order to discover better solutions.

The FC Portugal team has implemented some optimization algorithms till date, with the most recent being the following:

4.1 Genetic Algorithms

As the name states, this algorithm is a search heuristic inspired by the biological evolution, and, as such, it belongs to the class of evolutionary algorithms. Basically, it is an iterative process which generates a new population of individuals (chromosomes) in each iteration, also called generation. Each individual is a set of parameters (genes) representing a possible solution. The algorithm starts with a population of individuals, generated randomly across the solution space or seeded in a specific area. For each generation each individual is evaluated using a function that returns a fitness value.

From there, the next generations are produced based on the current one by applying some operators.

Selection

In each iteration some individuals are selected to be parents for the next generation. This selection is done randomly, with each individual having a probability of being chosen proportional with its fitness (fitter solutions have greater probability).

Crossover

This operator is analogous to the biological reproduction where two parents generate a child. Using some crossover technique, the child genes (parameters) are a combination of the ones from its parents.

Mutation

Analogous to biology mutation, it applies some mutation function to the children parameters, given a mutation probability.

Elitism

Carry the best individuals to the next generation without alteration to guarantee that the solution quality will not decrease during the iterations.

After each iteration, the next generation has normally better average fitness than the previous one, as the most fitter individuals have more probability to generate a child. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. Furthermore, the techniques to be used and parameters like the percentage of mutation, selection, etc has to be adjusted depending on the context to be applied, resulting in very different times of convergence and best solution fitness. A example pseudo code is showed in the figure 4.1.

```
Inputs: size α of population, rate β of elitism, and
rate γ of mutation and number δ of iterations
Output: solution X

//Initialization
generate α feasible solutions randomly;
save them in the population Pop;
//Loop until the teminal condition
for i = 1 to δ do
      //Elitism based selection
      number of elitism ne = α.β;
      select the best ne solutions in Pop and save then in Pop_1;
      //Crossover
      number of crossover nc = (α − ne)/2;
      for j = 1 to nc do
          randomly select two solution X_A and X_B from Pop;
          generate X_C and X_D by crossover to X_A and
X_B;
          save X_C and X_D to Pop_2;
      endfor
      //Mutation
      for i = 1 to nc do
          select a solution X_j from Pop_2;
          mutate X_j under the rate γ and generate a
new solution X'_j;
          if X'_k is unfeasible
              update X'_k with a feasible solution by repairing X'_j;
          endif
          update X_j with X'_j in Pop_2;
      endfor
      //Updating
      update Pop = Pop_1 + Pop_2;
endfor
//Returning the best solution
return the best solution X in Pop;
```

Figure 4.1: Pseudo code version of GA algorithm from [37].

4.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based stochastic method used in continuous and discrete optimization problems. In it, these called particles, where their position represent a candidate solution, that move iteratively in the search space trying to get to better positions by changing their velocities. These particles move towards their best known position and the swarm's best known position. In each iteration, improved positions may be discovered and override the previous ones which then take place on guiding the swarm, in hope to find a satisfactory solution for the problem.

For evaluating each candidate solution a cost function f is need with the objective of minimizing it. This function will return a real number for each candidate solution, representing its cost. The goal of the algorithm is to find a solution a for which $f(a) \leq f(b)$ for all b in the search-space, which would mean a is the global minimum.

The candidate solutions are particles ($\mathrm{P} = p_1, p_2, \ldots, p_k$) in a swarm where there are neighbourhood($Ni \subseteq P$) relations between them are represented as a graph G={V,E}. V is a vertex representing a particle and E is a edge representing a neighbour relation. Furthermore, at a specific time step t, p_i has a position $\vec{x}_i^t$, a velocity $\vec{v}_i^t$ and the best position(particle's personal best) it visited, represented by $\vec{b}_i^t$.

A pseudo code example is presented in follow image 4.2.

Inputs: Objective function $f : \Theta \rightarrow \mathbb{R}$, the initialization domain $\Theta' \subseteq \Theta$, the number of particles $|\mathcal{P}| = k$, the parameters w , φ^1, φ^2 , and the
stopping criterion S
Output: Best solution found

```
// Initialization
 Set t := 0
 for i := 1 to k do
    Initialize N_i to a subset of P according to the desired topology
    Initialize x_i^t randomly within Θ'
    Initialize v_i^t to zero or a small random value
    Set b_i^t = x_i^t
 end for

 // Main loop
 while S is not satisfied do
    // Velocity and position update loop
    for i := 1 to k do
       Set l_i^t := argmin_{b_j^t ∈ Θ | p_j ∈ N_i} f(b_j^t)
       Generate random matrices U_1^t and U_2^t
       Set v_i^{t+1} := w v_i^t + φ1 U_1^t (b_i^t − x_i^t) + φ2 U_2^t (l_i^t − x_i^t)
       Set x_i^{t+1} := x_i^t + v_i^{t+1}
    end for

    // Solution update loop
    for i := 1 to k do
       if f(x_i^t) < f(b_i^t)
           Set b_i^t := x_i^t
       end if
    end for
    Set t := t + 1
 end while
```

Figure 4.2: Pseudocode version of the standard PSO algorithm. [38].

4.3 CMA-ES

Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) is an evolutionary algorithm for non-linear, non-convex black-box optimisation problems in continuous domain. Typically it s applied to unconstrained or bounded constraint optimization problems, and search space dimensions between three and a hundred. Furthermore, it should only be applied when derivative based methods fail due to a rugged search landscape (discontinuities, sharp bends or peaks, noise, local optima, outliers) since they are usually faster.

As a evolution strategy, the candidate solutions are sampled according to a multivariate normal distribution in the $\mathbb{R}^n$, in which the pairwise dependencies between the variables are represented by a covariance matrix. This matrix is updated in each iteration, corresponding to a second order approach to a positive definite matrix. Thus it tries to learn the second order model of the underlying objective function, similar to the approximation of the inverse Hessian matrix in the Quasi-Newton methods.

The algorithm does not need much parameter tuning from the user as the strategy parameters are considered as part of the algorithm design. As such, the user only need to provide a initial solution, an initial standard deviation (step-size) σ, and an optional termination criteria [39] [40].
A simple pseudo of the algorithm is shown in figure 4.3.

```
set λ // number of samples per iteration, at least two, generally > 4
// initialize state variables
initialize m, σ, C = I, p_σ = 0, p_c = 0
while not terminate  // iterate
   // sample λ new solutions and evaluate them
    for i in {1...λ}
        x_i = sample_multivariate_normal(mean=m,covariance_matrix=σ^2 C)
        f_i = fitness(x_i)
   // sort solutions
    x_{1...λ} ← x_{s(1)...(λ)} with s(i) = argsort(f_{1...λ}, i)
    m' = m  // we need later m − m' and x_i − m'
   // move mean to better solutions
    m ← update_m(x_1, ..., x_λ)
   // update isotropic evolution path
    p_σ ← update_ps(p_σ, σ^{-1} C^{-1/2} (m − m'))
   // update anisotropic evolution path
    p_c ← update_pc(p_c, σ^{-1}(m − m'), ||p_σ||)
   // update covariance matrix
    C ← update_C(C, p_c, (x_1 − m' )/σ,..., (x_λ − m')/σ)
   //update step-size using isotropic path length
    σ ← update_sigma(σ, ||p_σ||)
return m or x_1
```

Figure 4.3: Pseudocode version of the CMA-ES algorithm. [41].

where

- $m \in \mathbb{R}^n$ is the distribution mean and current favorite solution
- $p_\sigma \in \mathbb{R}^n, p_c \in \mathbb{R}^n$ are two evolution paths (isotropic and anisotropic correspondingly), initially set to the zero vector
- $\sigma > 0$ is the step-size
- C is the a symmetric and positive definite $n \times n$ covariance matrix initialized with the identity matrix

chapter 5

Behaviours and Agent Improvement

The first steps taken in the code were some changes that could help achieve better results running the behaviours. Firstly, the sync mode was implemented to get faster simulations, then the gyroscope data was used for improving the agent known posture and finally, preconditions were added in the Slot Behaviours so they could reduce the execution in case of probable failure.

5.1 Improving Posture Estimate

During the course of the game, the agent must keep a model of the world state, and update it when possible so it knowns how to act in a given situation. One important information for the agent is its vertical lateral and frontal inclination so it knows if it is up, falling or is on the ground. For this, the vision is used to estimate the agent position using field flags, and from there is calculated the Torso reference frame vectors $Torso_x, Torso_y, Torso_z$.

Before, the frontal vertical inclination, as being calculated using the direction of the vector with the components z and x of the unity vector $TorsoZ$ 5.1. On the other hand, the lateral vertical inclination used the components z and y of the same vector 5.2.

$$vertInclinFront = Vector(TorsoZ_z, TorsoZ_x).getDirection() \tag{5.1}$$

$$vertInclinSide = Vector(TorsoZ_z, TorsoZ_y).getDirection() \tag{5.2}$$

5.1.1 Euler angles

Using the unit vectors $Torso_x, Torso_y, Torso_z$, which correspond to a reference frame, it is possible to calculate the euler angles [42] pitch 5.3 and roll 5.4, which are equivalent to vertical inclination frontal and lateral , respectively.

$$vertInclinFront = rad2degree(atan2(TorsoX_z, TorsoZ_z)) \tag{5.3}$$

$$vertInclinSide = rad2degree(atan2(-TorsoY.z, \sqrt{TorsoXz^2 + TorsoZ_z^2})) \tag{5.4}$$

As we can see in 5.1 the pitch error in the old method is large when the agent falls laterally. Besides that, the roll error is large when the agent falls on its back 5.2.

Furthermore, the correlation difference using the old method and using euler angles is: ***0.62495->0.83339*** in the pitch and ***0.70878->0.99945*** in the roll.

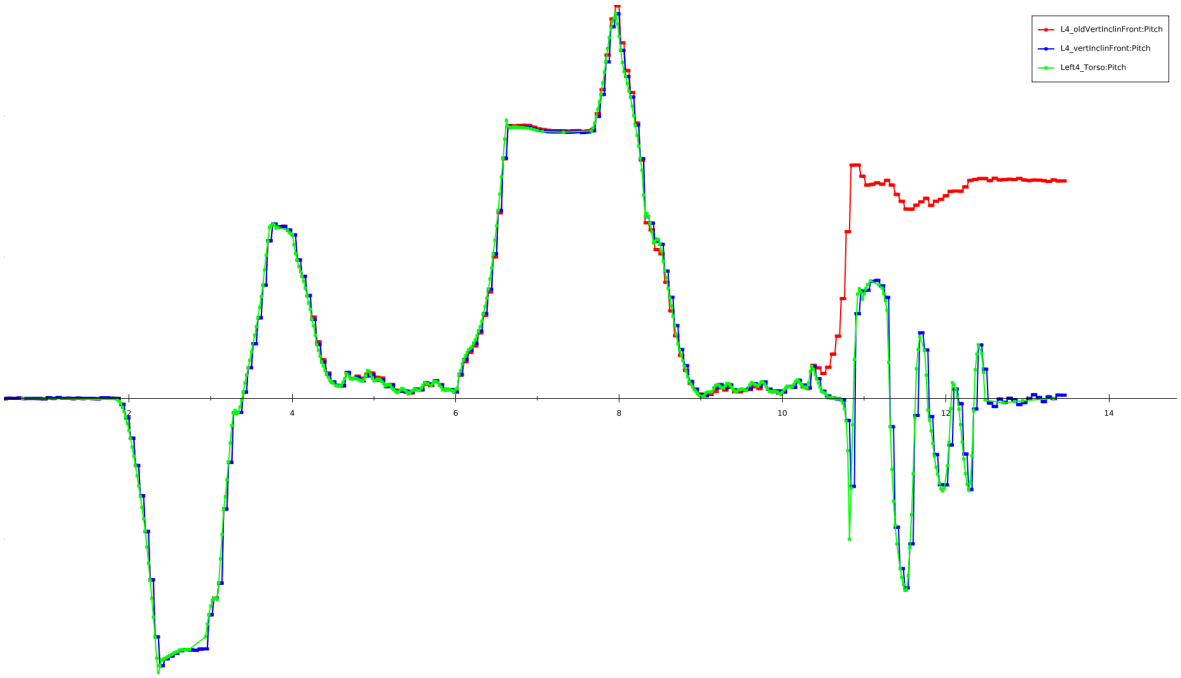


Figure 5.1: Comparison between vertical inclination front using the old method versus computing pitch. ***Green*** is the ground truth from the server, ***red*** is using the old method and ***blue*** is the computed pitch.

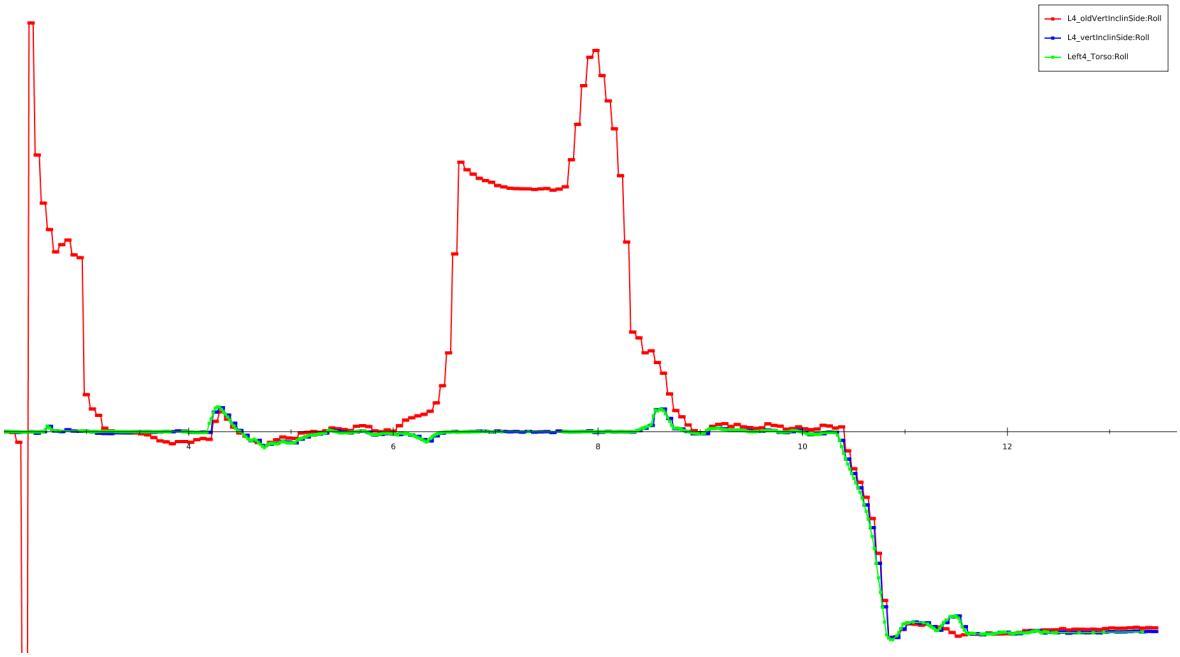


Figure 5.2: Comparison between lateral vertical inclination using old method versus computing roll. ***Green*** is the ground truth from the server, ***red*** is using the old method and ***blue*** is the computed roll.

5.1.2 Gyroscope

One problem is that this perceptor is only available every third cycle, so there are a cycles where the agent does not update its inclination data. In order to resolve this issue, it is used of the gyroscope sensor, which is received every cycle. In the cycles where the vision is not available, the gyro rate received in degrees per second is integrated between the cycle duration and added to the current values of the vertical inclination frontal 5.5 and lateral 5.6 (agent pitch and roll, respectively).

$$vertInclinFront- = GyroRate_x * CYCLE\_DURATION\_S \tag{5.5}$$

$$vertInclinSide+ = GyroRate_y * CYCLE\_DURATION\_S \tag{5.6}$$

With the use of gyroscope, the vertical inclination gets closer to the ground truth. As we can see in 5.3 and 5.4, the gyroscope removes that ladder effect resulted from the cycles in which the vertical inclination is not updated. Furthermore, the correlation difference using only euler angles and using gyroscope is: ***0.83339->0.83560*** in the pitch and ***0.99945->0.99992*** in the roll.

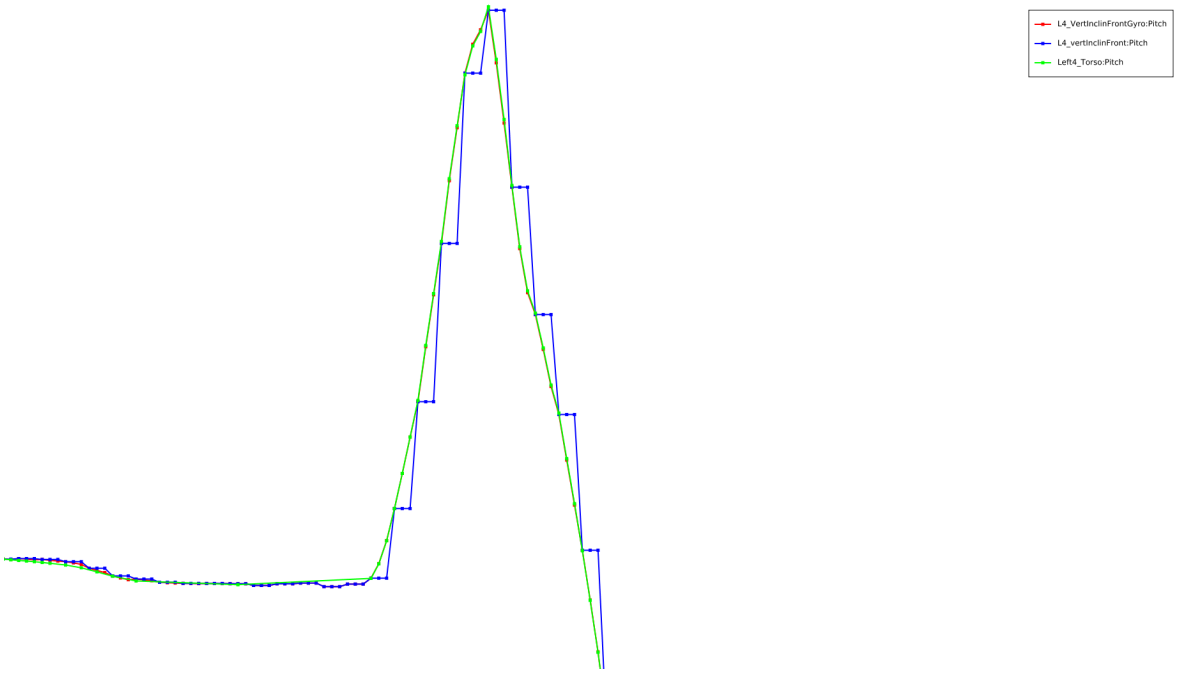


Figure 5.3: Comparison between only computing pitch versus using gyroscope. ***Green*** is the ground truth from the server, ***blue*** is only using euler angles and ***red*** using also the gyroscope.

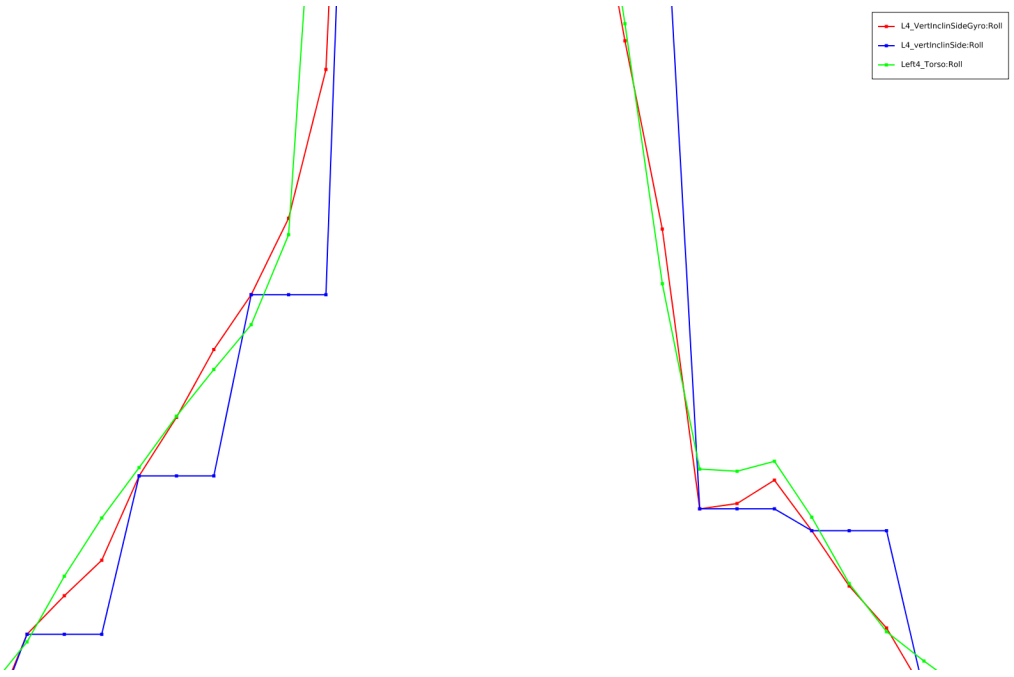


Figure 5.4: Comparison between only computing roll versus using gyroscope. ***Green*** is the ground truth from the server, ***blue*** is only using euler angles and ***red*** using also the gyroscope.

5.1.3 Agent Posture State

With better values of the vertical and lateral inclination, a state for the agent was implemented, so the developer gets a better understanding of the agent state, to simplify the conditions made in the code and so the agent can react depending on the current situation. The various sates are:

UP
: The agent is up and can execute the normal game behaviours.

GETTING_UP
: The agent is executing a get up behaviour. While is not in successfully in *UP* state it must not execute any game behaviour such as walk and kick.

FALLING_FRONT, FALLING_BACK, FALLING_SIDE_LEFT, FALLING_SIDE_RIGHT
: The agent as passed a point of no return, where it cannot recover its balance. So it goes to *Zero Position* behaviour, which reset all the joints positions to a standard start joint configuration, so when it hits the ground, a get up behaviour starts as fast as it can.

GROUND_BACK, GROUND_CHEST, GROUND_SIDE_LEFT, GROUND_SIDE_RIGHT
: The agent is fallen on the ground. It calls the appropriate get up behaviour.

5.2 Preconditions

The simulation environment is very dynamic and has many conditions that are difficult to manage and to foresee. One behaviour could function well in one situation and fail in some other. In behaviours that are not adaptable, they will run every time till they reach the finished condition, as nothing exterior happened. For example, when a agent is in the ground and is trying to getup, if it is pushed, the getup behaviour is executed till the end. This leads to a loss of time that can be critical in the game, for example, in a ball dispute.

With this in mind, a solution was implemented based on preconditions and applied to Slot Behaviours. They are defined in XML files, where a behaviour is declared divided in various slots. In the beginning of the execution of each slot, the preconditions declared are verified. If they pass, the behaviour executes the corresponding slot, if not the behaviour considers that it must end and it activates its finished flag. Till now, the parameters accepted are:

incl_front:gt
: Frontal inclination greater than

incl_front:lt
: Frontal inclination less than

incl_side:gt
: Lateral inclination greater than

incl_side:lt
: Lateral inclination less than

incl_front:outer
: Frontal inclination Outer interval

incl_front:inner
: Frontal inclination Inner interval

incl_side:outer
: Lateral inclination Outer interval

incl_side:inner
: Lateral inclination Inner interval

incl_front:abs_gt
: Absolute Frontal inclination greater than

incl_front:abs_lt
: Absolute Frontal inclination less than

incl_side:abs_gt
: Absolute lateral inclination greater than

incl_side:abs_lt
: Absolute lateral inclination less than

z_position:gt
: Z position of the agent greater than

z_position:lt
: Z position of the agent less than

As we can see in 5.5 there are conditions in the first slot *incl_front:abs_lt="10"* *incl_side:abs_lt="10"*, that must be meet for the slot to execute.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE joints [
<!ENTITY head1  "0"  >
...
<!ENTITY rarm4  "21"  >
]>

<behavior name="FallBack" type="SlotBehavior">

  <slot name="ForceFalling" delta="0.5" incl_front:abs_lt="10"
   ↪  incl_side:abs_lt="10">
    <move id="&head1;" angle="0"    />
    <move id="&head2;" angle="0"    />
    <move id="&lleg1;" angle="0"    />
    <move id="&rleg1;" angle="0"    />
    <move id="&lleg2;" angle="0"    />
    <move id="&rleg2;" angle="0"    />
    <move id="&lleg3;" angle="0"    />
    <move id="&rleg3;" angle="0"    />
    <move id="&lleg4;" angle="0"    />
    <move id="&rleg4;" angle="0"    />
    <move id="&lleg5;" angle="-45"    />
    <move id="&rleg5;" angle="-45"    />
    <move id="&lleg6;" angle="0"    />
    <move id="&rleg6;" angle="0"    />
    <move id="&larm1;" angle="-90"  />
    <move id="&rarm1;" angle="-90"  />
    <move id="&larm2;" angle="0"   />
    <move id="&rarm2;" angle="0"  />
    <move id="&larm3;" angle="0"  />
    <move id="&rarm3;" angle="0"   />
    <move id="&larm4;" angle="0"    />
    <move id="&rarm4;" angle="0"    />
  </slot>
  <slot name="ResetPositionAndWait" delta="0.5" incl_front:gt="80"
   ↪  incl_front:lt="100">
    <move id="&lleg5;" angle="0"    />
    <move id="&rleg5;" angle="0"    />
  </slot>
</behavior>
```

Figure 5.5: Preconditions example

chapter 6

Optimization Process Improvement

In the FC Portugal code, there are already some optimization algorithms implemented such as Genetic Algorithms, PSO and CMA-ES. In this thesis the optimizations were performed using CMA-ES. Still, there was some room to improve and automatize the optimization process. The next sections described what was done in that sense.

6.1 Sync Mode

According to the competition rules the server runs in real-time, which means its cycles have a duration of 20 ms, in which the server is listening for the agent effector commands. This creates two problems. The agent has limited time to process and send its commands within the cycle time, if it ca not, the commands will only be applied in the next cycles, which worsen the expected behaviour. The other problem is the usage of the resources that are not used to the maximum, since the server waits till the end of the cycle to process, even if it has already received all the agent commands. For the competition, this problems makes sense as the cycle time provides a just processing time restriction to all the teams. But, for developing and optimization purposes, it is way better to run the simulation as fast as the CPU can. This way, the server only waits for the agent commands and a synchronise message which signals the end of the agent cycle. After it receives all the agents Sync message, the server processes all the commands and proceeds to the next cycle. In addition to simulation speed time improvement, it can also be used to detect strange cycle times from the agents.

6.1.1 Sync Mode Usage

Before this sync mode implementation, for each command the agent generated, one message was sent to the server. This approach was not working when transiting to sync mode. To exploit this mode, the server connection code in the agent was changed so all messages from the same cycle are now aggregated and for each message sent by the agent, a synchronize was added. Since then, the agent code can support sync mode. If the server is not running in agent sync mode, the synchronize messages

are ignored. Finally, to configure the server in sync mode, *enableRealTimeMode* variable in the file */usr/local/share/rcssserver3d/rcssserver3d.rb* file must be set to *false* and *agentSyncMode* in */home/$USER/.simspark/spark.rb* to *true*. To simplify changing between real time mode sync mode, a python script was created, which can be called as following:

./syncMode <true|false>

As a result the time a simulation takes is drastically decreased. Without sync mode, each cycle takes 20 ms and each part takes 5 minutes, regardless the number of agents in game. With sync mode, the simulations runs as fast as the computer can, so the number of agents in game affect the simulation time. In a laptop with few agents the simulation time is drastically decreased, but with many agents, the difference between running in sync and real mode it is not visible. On the other hand, in a desktop, even when running with all the 22 agents, the simulation time is greatly reduced.

6.2 Constraint Free Environment

The normal simulation execution is the same as during the competition games. Because of that, there are some constraints such as: the game has limited time, 5 minutes for each part; the agents can not move the ball and cannot beam themselves after the game have started. This affect badly the optimization process because it needs to run for as long as needed without human intervention.

For that purpose, a constraint free environment was created as explicated below.

6.2.1 Solution

Step 1. The first step was to create a patch for the server code. It is a simple modification in the *rcssserver3d* code so the playmode can be calculated based on the existence of a Linux environment variable named *ALWAYS_PLAYON*. When this variable exists and is set to *1*, the server is always in *PlayOn* mode, otherwise the server runs normally. The *PlayOn* mode is used as the normal state of the game when the ball is running. With *PlayOn* mode always active the game never ends. After the application of this patch, the server code needs to be compiled and installed. To set the environment variable, *"export ALWAYS_PLAYON=1"* must be run in the console before calling the server executable, or defined in a *∼/.profile* file or similar Linux file.

Step 2. According to the game rules, the agent can only beam itself during the kickoff, and the ball only can be moved after the kickoff. To overcome this, it was created a trainer proxy (see image 6.1), which allows the agent to call all the trainer commands (see section 2.2). The most useful trainer commands are those to control the robot and the ball, imperative to run kick behaviour optimizations. There was a tentative of connecting the agents directly to the server using the monitors port, but the connections were failing, so a proxy was needed. The proxy allows connections from the agents and connects itself to the server as a monitor. By default, it listen for agent incoming connections using port 3300.

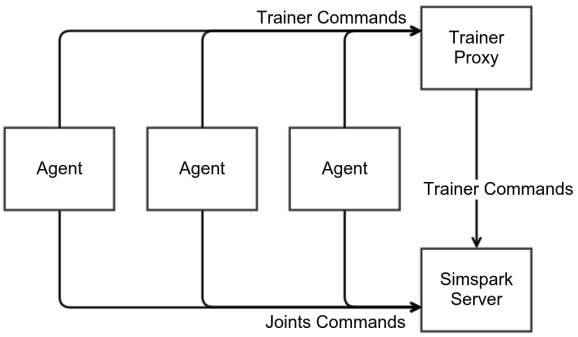


Figure 6.1: Trainer Proxy

./trainerProxy.py -pp <proxyport> -s <server_port>

proxy_port

Port number to listen for agent incoming connections.(default=3300)

server_port

Port number which rcssserver3d exposes for monitor connections (default=3200).

Step 3. Finally, the agent used for optimization purposes has to connect to both the server and the proxy. A command line option is used to specify the port number. There is a argument called ***pp*** *<proxy_port>*. For example, if the proxy uses the port 3400:

./fcpagent -pp 3400

6.3 Optimization Agent

Normally the optimization process of finding good value sets consumes a lot of time and resources, as it needs to evaluate a lot of possible solutions, and for each solution, resample multiple times to average each individual cost value. Before, the process of optimization was being done running only one agent for server to optimize one behaviour each time. In order to improve this, a new executable was created so multiple agents can run in the same server (maximum of 22 agents for server), running different optimizations with the same executable.

This executable was divided in multiple scenarios (GETUP, KICK and WALK), each one with its parameters and its behaviours.Currently any XML defined behaviour is supported for GETUP and KICK. This scenarios execute the behaviour passed as argument and each learning scenario has specific sequence and cost or fitness function.

GETUP

Scenario where the getup behaviours can be evaluated. This will execute a fall behaviour, followed by the getup behaviour. It has a cost function associated with the time it takes to execute. If, at the end of the getup behaviour, the agent is not up, the cost function is affected greatly to reflect the error. Furthermore, the fall type and the getup file path need to be passed as argument.

KICK

Scenario where kick behaviours can be optimized. The ball will be positioned in front of the agent and then the kick behaviour is executed. The fitness function is related to the distance

that the ball travels and if it goes straight forward. This fitness value is affected greatly if the ball does not move, indicating that the agent missed it. The behaviour file path need to be passed as argument.

WALK

Scenario used for optimizing the walk behaviour. Currently, it executes the ZMP walk in the goal direction. The fitness function evaluates the distance covered during a limited defined time and its deviation from the target position (goal). Furthermore, this fitness is affected greatly if the agent falls.

The optimization agent needs to know what parameters to optimize. In the slot behaviours this is done adding new attributes to the XML file, stating the parameters to be optimized. The new attributes have the same name as the attributes representing the parameters with the prefix *'o:'* added. The value of the new attributes is just an enumeration and can be used to specify that two or more parameters should have the same value during the optimization process. Figure 6.2 shows an example with several parameters to be optimized. For instance, lines *10* and *11* show two parameters whose values should be the same.

```
1  <?xml version='1.0' encoding='ISO-8859-1'?>
2  <!DOCTYPE behavior [
3  <!ENTITY head1 "0">
4  ...
5  <!ENTITY rarm4 "21">
6  ]>
7  <behavior name="r0_GetUpFront" type="SlotBehavior" correctAnglesOutOfRange="false">
8
9    <slot name="leftLegUp" delta="0.205311" o:delta="1">
10     <move id="&larm1;" angle="14.514300" o:angle="10"/>
11     <move id="&rarm1;" angle="14.514300" o:angle="10"/>
12     <move id="&larm4;" angle="0"/>
13     <move id="&rarm4;" angle="0"/>
14     <move id="&lleg3;" angle="0"/>
15     <move id="&rleg3;" angle="0"/>
16     <move id="&lleg4;" angle="0"/>
17     <move id="&rleg4;" angle="0"/>
18     <move id="&lleg5;" angle="0"/>
19     <move id="&rleg5;" angle="0"/>
20     <move id="&lleg5;" angle="0"/>
21     <move id="&rleg5;" angle="0"/>
22   </slot>
23
24   <slot name="leftLegUp" delta="0.079244" o:delta="2">
25     <move id="&lleg3;" angle="259.929000" o:angle="11"/>
26     <move id="&rleg3;" angle="259.929000" o:angle="11"/>
27     <move id="&lleg4;" angle="-133.612000" o:angle="12"/>
28     <move id="&rleg4;" angle="-133.612000" o:angle="12"/>
29     <move id="&lleg5;" angle="124.630000" o:angle="13"/>
30     <move id="&rleg5;" angle="124.630000" o:angle="13"/>
31   </slot>
32 ...
33 </behavior>
```

Figure 6.2: Example of XML behaviour file to optimize

The current version of the optimization agent accepts the following configuration parameters:

-r <robotType>

Heterogeneous model type to use.

-scenario <Getup,Kick,Walk> (default is Getup)

Optimization scenario to run.

-b x y z

Agent initial beam position. As the server may be running with multiple agents, their positions must be disperse so they do not interfere with each other optimization.

-opt <behaviourFile>

Path of the behaviour to optimize.

-fallType <None,FallFront,FallBack>

Fall behaviour to be used in the getup scenario.

-res

Number of resamples to run (default is 5).

-pop

Population (default is 20).

6.4 Update Optimization Parameters

The result of an optimization is a set of possible solutions, stored in a text file. The user must manually test some of them and choose one before update the working behaviour. A tool was developed to aid in the updating of a slot behaviour. The tool receives as arguments the file to be updated and the new parameter values.

The tool was implemented as a python script, which parses and updates the behaviour file. Every time an attribute with the prefix *'o_'* is found, the corresponding normal attribute is updated with the given new value. More details on the attributes format are in 6.3.

Requisites: Python2.7 python2-lxml

./updateOptParams.py [filename] [newValues]

Ex: *./updateOptParams.py movs/type0/getup/GetupFront.xml 0.2 0.3 0.4*

6.5 Heterogeneous Models

The simulation league has evolved since the use of a sphere in 2004 to represent a agent till the use of humanoid agents in 2007. In 2013, heterogeneous models were introduced and, in 2014, rules were made mandatory to force their use in the teams (at least three different model types must be used). In order to fulfill these new requirements, some changes had to be done in the agent code/project.

1. In the FC Portugal project there exists a XML file describing the robot model used. It describes the body parts, joints information like its position, etc. To support heterogeneity, one file for each model type was used, with its custom information. The name of the file is *nao%d.xml*, replacing *%d* with the heterogeneous model number (0-4) . Furthermore, note the *"nao/nao_hetero.rsg 3"* in the beginning of *nao3.xml* 6.3 which is the string that is sent to the Simspark server so it knowns which model to use.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<agentmodel name="nao" type="humanoid" rsgfile="nao/nao_hetero.rsg 3">

<bodypart name="head" mass="0.35" />
<bodypart name="neck" mass="0.05" />
<bodypart name="torso" mass="1.2171" />
<bodypart name="lshoulder" mass="0.07" />
<bodypart name="rshoulder" mass="0.07" />
<bodypart name="lupperarm" mass="0.15" />
<bodypart name="rupperarm" mass="0.15" />
<bodypart name="lelbow" mass="0.035" />
<bodypart name="relbow" mass="0.035" />
<bodypart name="llowerarm" mass="0.2" />
<bodypart name="rlowerarm" mass="0.2" />
<bodypart name="lhip1" mass="0.09" />
<bodypart name="rhip1" mass="0.09" />
<bodypart name="lhip2" mass="0.125" />
<bodypart name="rhip2" mass="0.125" />
<bodypart name="lthigh" mass="0.275" />
<bodypart name="rthigh" mass="0.275" />
<bodypart name="lshank" mass="0.225" />
<bodypart name="rshank" mass="0.225" />
<bodypart name="lankle" mass="0.125" />
<bodypart name="rankle" mass="0.125" />
<bodypart name="lfoot" mass="0.2" />
<bodypart name="rfoot" mass="0.2" />

<!-- joint 0 -->
<joint name="head1" perceptor="hj1" effector="he1" xaxis="0" yaxis="0"
 ↪  zaxis="-1" min="-120" max="120">
<anchor index="0" part="neck"  x="0" y="0" z="0.0" />
<anchor index="1" part="torso" x="0" y="0" z="0.09" />
</joint>
...
</agentmodel>
```

Figure 6.3: Initial part of nao3.xml file

2. In the code, the walk parameters and mid level skills had to be tuned for each model type.

3. At last, new behaviours for each heterogeneous type were created. First, it was used the standard model behaviours, then they were manually tuned for each type, so they behaved properly. After that, each used behaviour defined in a XML files was optimized described in the Optimization Results section 6.6.

6.6 Results

The process of optimization was automated to some extent.

It envolves 4 applications: the optimization agent, the trainer proxy, the 3D soccer server, and a matlab program running the CMA-ES algorithm. After connection, the optimization agent sends the required optimization data to the matlab program, that, afterwards, uses the agent as a server to compute the cost function of the individuals. By its turn, the agent, in order to compute the value of the cost function, runs a simulation in the soccer server, and returns the computed value to the matlab program. These interactions last until the optimization process ends. Figure 6.4 illustrate the interaction among the different applications.

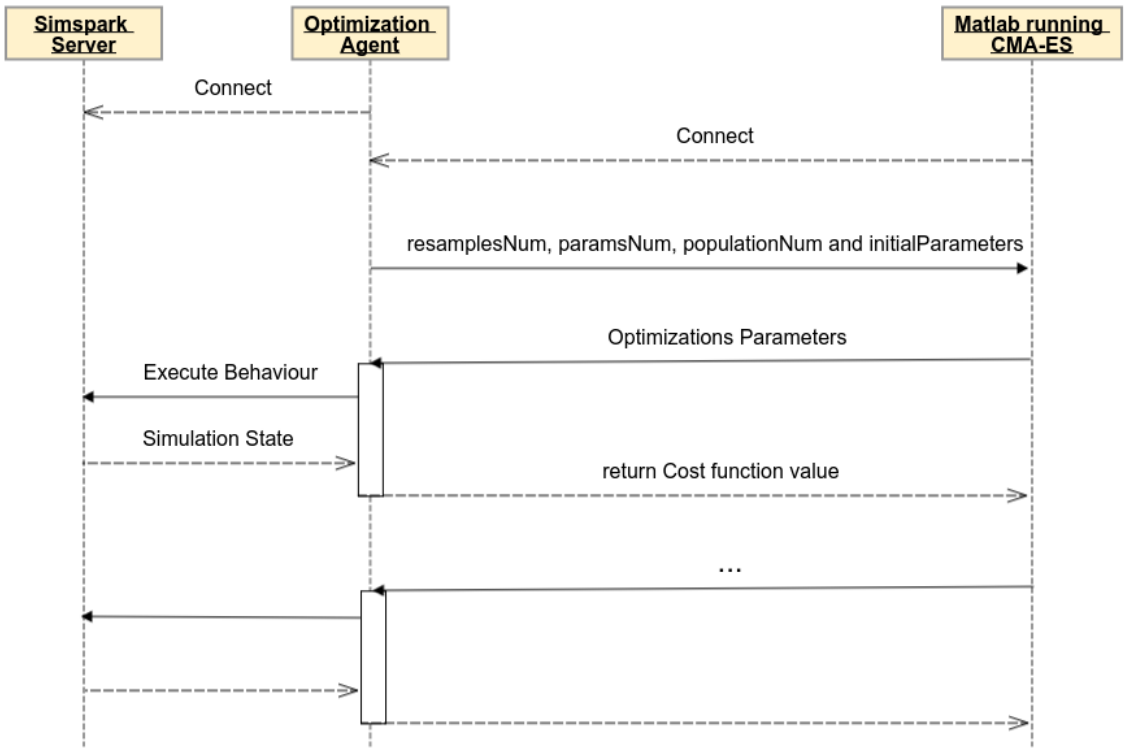


Figure 6.4: Optimization Flow

Some decisions were taken in how to configure the optimizations that were realized. Not all the robot joints were managed to be manipulated by the optimization algorithm, as it could be a very long process and unnecessary. It was decided to consider only the most important joints for each behaviour.

Different runs of the simulation of a behaviour, for the same initial conditions, can lead to different results. So, a single run should be avoided to calculate the cost function. It was decided to do several runs and calculate the average of the values. However, the number of runs has a great influence in the optimization process time. It was decided to use 5 runs. With 3 evaluations, it was verified that the behaviours were not consistent, and using more than 5 evaluations is just in cases where we want really consistency, which, on the other hand, affect the possible achievable fitness. The population used was 30 because it is not so small to be a local search, neither is to big to be a global search.

As a result of this tuning, better times were obtained for the behaviours optimized as shown below. The robots type 1 and 3 were not optimized because they are not currently used in the team heterogeneous setup.

For the getups, the Getup scenario was used, being obtained the following values 6.1 6.2 (in seconds).

	Model Type		
	0	2	4
Before	2.6	2.6	2.6
After	1.12	0.74	0.98

Table 6.1: GetUpBack

	Model Type		
	0	2	4
Before	1.42	1.42	1.41
After	0.74	0.68	0.72

Table 6.2: GetUpFront

chapter 7

Development and Debugging Tools

In any software project, the development and debugging processes/tools are critical to quickly pass from an idea into a fully working and robust solution. Furthermore, the understanding of what is happening in the simulation is greatly increased. It became clear that the project could make good use of new tools. So, three tools were developed. The first is a monitoring interface for the agents where, in runtime, it is possible to see code variables, input data and control the cycle time. The second is a post simulation analyser which uses the logs from the agents and the server as input and then, parsing and filtering the data, it is possible to export in multiple formats, view plots of the data, etc. Finally, the last one is a tester for behaviours defined in XML files which gives, in runtime, the possibility to edit behaviours, run a sequence of behaviours, change the robot heterogeneous type, etc.

They are detailed in the following sections.

7.1 Agent Monitor Window

Before, the analysis of each code change was being done by visualizing the result in the simulation monitor, then, in the end of the simulation, analysing the log containing the standard output from the executable. This log normally contain thousands of lines just for one agent, so, to simply search for a game situation and understanding what is happening was quite difficult. Furthermore, during the execution of a simulation it becomes greatly useful to have the possibility to see the values from the code variables as they change in each situation, or other debug information.

The solution found was to use a graphical interface to monitor program variables specified by the user through an API. Each agent can have its own windows. Thus, in order to allow all windows to be seen at the same time, their position on the screen depends on the id of the agent. Figure 7.1 shows the 11 windows of 11 agents, partially overlapped by the server window.

Figure 7.1: Agent Monitor Window running 11 agents, each with his window.

The interface was implemented in GIMP Toolkit (GTK) [43] and runs in a separated thread. To use the tool one must use the parameter `-s`.

Ex:./fcpagent -s

Requisites: >=libboost1.48-all-dev (Ubuntu), >=boost 1.48.0 (Arch Linux)

7.1.1 Developer API

During development, it is often necessary to change the values of given internal variables and see the effect of such changes. Being possible to do this without recompiling the program is a major advantage. The API available to the developer/programmer, in addition to the possible visualization of internal variables, allows for the change of given variables. The changes are done though the use of three different types of inputs, associated with callbacks. When a changed event is detected in a registered input, the corresponding callback will be triggered, and, from there, the new value is processed by the developer.

Show variable

The main feature is to give the ability to show any value/string and update it during the course of the simulation. Giving a variable name and its value, one can call the following functions 7.2:

```
void showVariable(string variableName, float value);
void showVariable(string variableName, string value);
```

Figure 7.2: Agent Monitor Window show variables

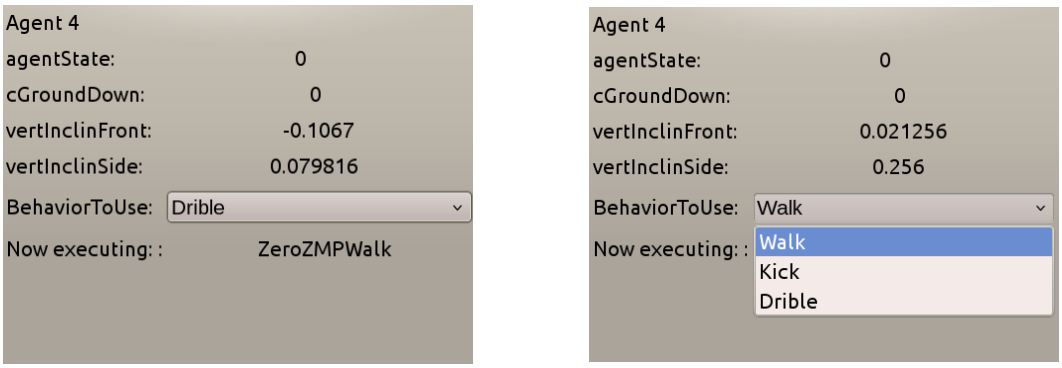
Register *ComboBox*

Registering a *ComboBox* gives a set of possible selectable values (see figures 7.4a and 7.4b) that

can be used for example, for controlling which behaviour should the agent execute next. The register function is presented in the figure 7.3.

```
void registerComboBox(string comboBoxName, vector<string> values, int active, void*
 ↪  changedCallback);
```

Figure 7.3: Register Combo Box in Agent Monitor Window



(a) ComboBox Closed (b) ComboBox Opened

Figure 7.4: Agent Monitor Window with ComboBox

Register *SpinButton*

Registering a *SpinButton* (see figure 7.6) let the developer to write any string to be interpreted by the callback he wrote. One possible use is to test a parameter or multiple ones to achieve the expected behaviour. The register function is presented in the figure 7.5.

```
void registerSpinButton(string spinButtonName, GtkAdjustment* spinnerAdjustment,
 ↪  void* valueChangedCallback);
```

Figure 7.5: Register Spin Button in Agent Monitor Window

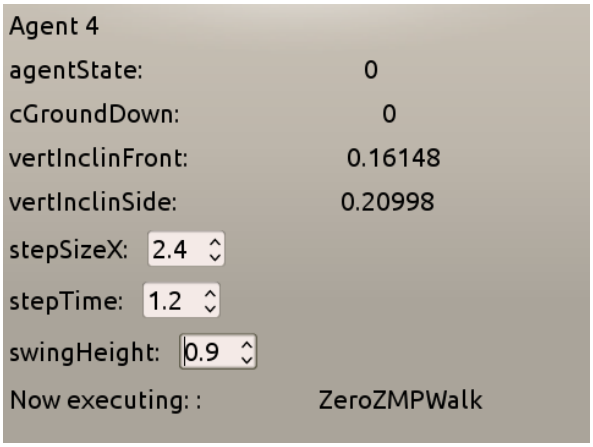


Figure 7.6: Agent Monitor Window with SpinButton

Register *keyboard key*

Registering a ***keyboard key*** may be used, for example, to control an agent dribbling direction, testing different angles and situations. The register function is presented in the figure 7.7.

```
void registerKeyPressed(int keyValue, gboolean (*keyPressedCallback)(GtkWidget*,
 ↪  GdkEventKey*, gpointer));
```

Figure 7.7: Register keyboard key in Agent Monitor Window

7.1.2 Controlling the Cycle Time

As the simulation cycle is 20 ms in real mode, it is humanly impossible to see the variation of one or more variables in each cycle. To give the developer total time control of the simulation, server sync mode was explored. The server, running in sync mode, will always wait for all agents' synchronize messages. Controlling the sending time of one agent synchronize message, one can control all the simulation cycle time. To implement this, it was used the ***boost chrono*** library, available since version 1.47.0, which contains a high resolution clock. With this clock, it is measured the agent cycle time: the time between the agent receives a message from the server and finish processing it. From there, the agent waits until the cycle time the user sets in the interface, to send the synchronize message. Calling the ***-ss*** parameter in one agent, the Agent Monitor Window show the cycle time controls (see image 7.8), in which is possible to:

1. Pause simulation
2. Go to next simulation cycle
3. Change simulation cycle time

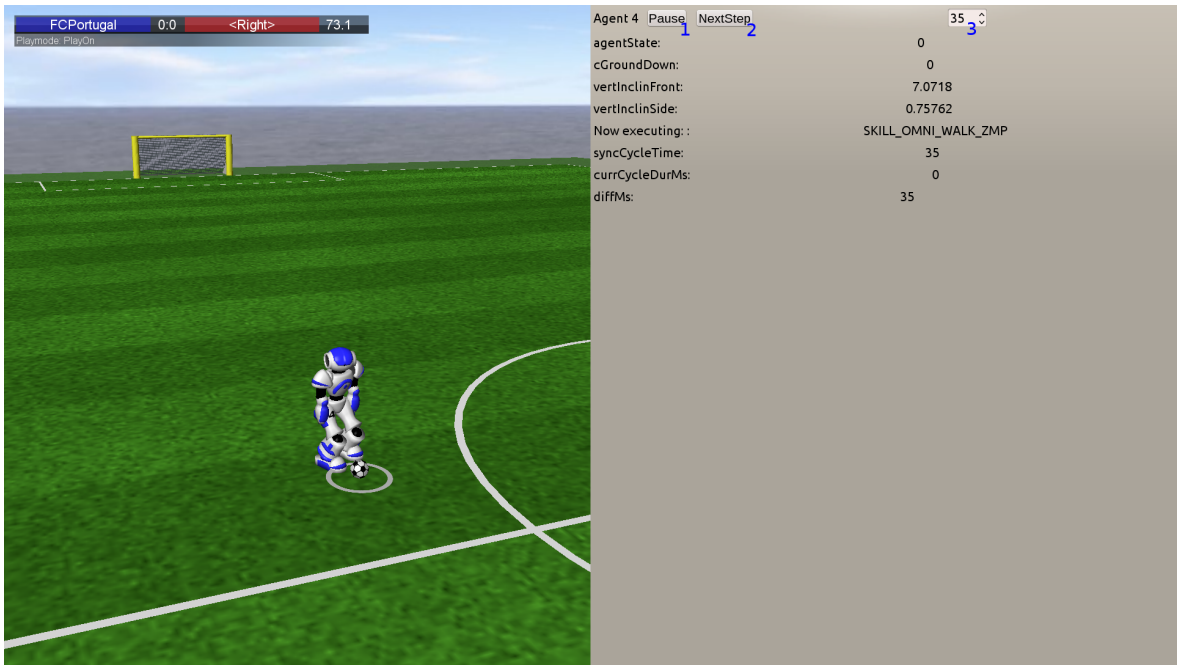


Figure 7.8: Agent Monitor Window with cycle time controls

Ex:./fcpagent -u 2 -ss

7.2 Log Analyser

Using the Agent Monitor Window, it is possible to view the simulation state in runtime. In other cases, a developer may want to use the resulting agent and server logs to do after-simulation analysis. So, a Graphical User Interface (GUI) tool was developed, increasing also the information obtained through the manipulation of the log files and the information visualization. Due to its simplicity to deal with strings and good documentation, Java was used for programming the GUI tool, with Swing [44] toolkit being used to display the interface elements.

7.2.1 Extracting Information

The input for the application is the logs generated from the agents and the server. In the server, the simulation must be configured to record a log file. In the *ruby* file in */usr/local/share/rcssserver3d/rcssserver3d.rb* or similar path, there is a variable in the beginning of the file named *recordLogfile* that must be set to *true*.

In each execution of the server, a *sparkmonitor.log* file containing all the information needed for a monitor to reproduce the simulation is generated. This file contain S-expressions using the protocol detailed in section 2.2.2.

On the other hand, the log files from the agents have their standard output from the execution of the code, without any standard format. So, to extract meaningful data, the program gives the user the possibility to declare regex expressions with groups in it. Each group as a meaning, which currently can be: the simulation time named *time*; a matrix position *Ny, Nz, Zero3, Ox, Oy, Oz, Zero7, Ax, Ay, Az, Zero11, Px, Py, Pz, One15*; an Euler angle *Pitch, Roll, Yaw*; or a custom variable *Val1, Val2, Val3, Val4*. These regex expressions are applied in the agent logs, extracting the groups data. An example expression can be *gameTime:(FLOAT) vertInclinFront:(FLOAT)* with the *Group 1* denoting the *Time* and *Group 2* as the *Pitch*. This expression will search for lines in the log file such as *gameTime:2.4 vertInclinFront:45.2*, extracting *2.4* and *45.2* as *Time* and *Pitch*, respectively.

7.2.2 Export Options

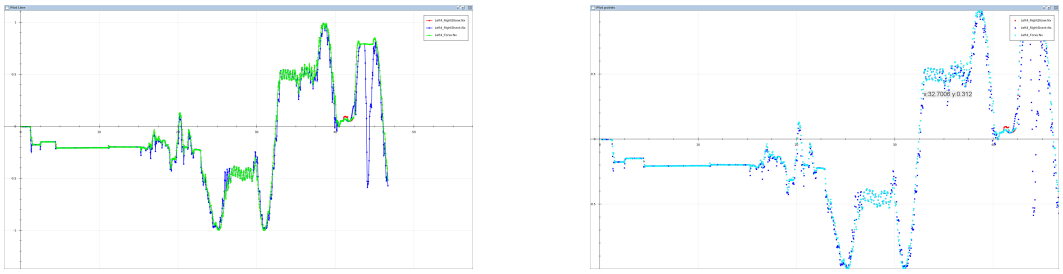
As output, the application has various options such as:

Log

The Log export option will generate a file with the same format as the *sparkmonitor.log* detailed in sub section2.2.2. The generated log can then be viewed in any simspark monitor. It may be helpful, for example, to put a 3D model of a ball in the positions that the agent thinks the ball was. So, in this case, two balls would appear in the screen, and the user could see the difference between the real position of the ball and the one it calculates from the vision system.

Plots

The Plots export option produces either line or point plots (see figure 7.9 for an example of each type). This is helpful to see the variation of a variable through the simulation, and/or to compare it with other variables.



(a) Lines Plot (b) Points Plot

Figure 7.9: Two types of plots available

XLS

The XLS export option produces a file like the figure 7.10.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Time	Nx	Ny	Nz	Zero3	Ox	Oy	Oz	Zero7	Ax	Ay	Az	Zero11	Px	Py	Pz	One15	Pitch
2	0.18	-0.00341	-0.99999	-2E-007	0	0.999994	-0.00341	7.2E-005	0	-7E-005	9.4E-008	1	0	-4.56328	1.19874	0.383706	1	-0.0041
3	0.38	-0.00341	-0.99999	-2E-007	0	0.999994	-0.00341	0.00011	0	-0.00011	2.0E-007	1	0	-4.56692	1.19875	0.37809	1	-0.006
4	0.46	-0.00341	-0.99999	-2E-007	0	0.999994	-0.00341	8.2E-005	0	-8E-005	6.6E-008	1	0	-4.56863	1.19876	0.372325	1	-0.004
5	0.54	-0.00341	-0.99999	-2E-007	0	0.999994	-0.00341	5.9E-005	0	-6E-005	-9E-009	1	0	-4.56977	1.19876	0.367041	1	-0.0033
6	1.06	-0.00341	-0.99999	-2E-007	0	0.99998	-0.00341	0.005293	0	-0.00529	1.8E-005	0.999986	0	-4.57002	1.19876	0.360814	1	-0.3032
7	1.14	-0.00404	-0.99999	-0.00054	0	0.999974	-0.00404	0.005888	0	-0.00589	-0.00051	0.999983	0	-4.56477	1.20291	0.356038	1	-0.3373
8	1.22	-0.00386	-0.99999	0.000259	0	0.999962	-0.00386	-0.00782	0	0.007816	0.000229	0.999969	0	-4.56358	1.21065	0.357247	1	0.44779
9	1.26	-0.01212	-0.99992	-0.00209	0	0.999027	-0.01202	-0.04244	0	0.042415	-0.0026	0.999097	0	-4.55948	1.21292	0.356629	1	2.43256
10	1.3	-0.05731	-0.99835	-0.00299	0	0.996298	-0.057	-0.06436	0	0.064082	-0.00667	0.997922	0	-4.55809	1.21312	0.356098	1	3.69001
11	1.34	-0.10391	-0.99458	-0.00376	0	0.991373	-0.10327	-0.08073	0	0.0799	-0.01212	0.996729	0	-4.55821	1.21067	0.35495	1	4.6303
12	1.38	-0.11629	-0.99321	-0.0019	0	0.989372	-0.11567	-0.08811	0	0.087289	-0.01213	0.996109	0	-4.55924	1.20598	0.34758	1	5.05471
13	1.42	-0.10887	-0.99402	0.008246	0	0.988855	-0.10915	-0.10126	0	0.101554	-0.00287	0.994826	0	-4.55834	1.20136	0.344957	1	5.81192
14	1.46	-0.14257	-0.98967	0.015311	0	0.983822	-0.14339	-0.10739	0	0.108472	-0.00025	0.994099	0	-4.55568	1.19839	0.348723	1	6.1653
15	1.5	-0.1668	-0.98589	0.01381	0	0.979611	-0.1673	-0.11123	0	0.111973	-0.00503	0.993699	0	-4.5536	1.19639	0.349186	1	6.38694
16	1.54	-0.1789	-0.98381	0.010368	0	0.977327	-0.17892	-0.11323	0	0.113251	-0.01012	0.993515	0	-4.55231	1.19641	0.345564	1	6.50183
17	1.58	-0.17723	-0.98417	-0.00205	0	0.977517	-0.17579	-0.11644	0	0.114234	-0.02264	0.993196	0	-4.55202	1.19569	0.350315	1	6.68658
18	1.62	-0.17638	-0.98426	-0.0109	0	0.97762	-0.17388	-0.11843	0	0.114671	-0.03154	0.992903	0	-4.55194	1.19464	0.351225	1	6.80184
19	1.7	-0.17679	-0.98416	-0.01333	0	0.977138	-0.17387	-0.12236	0	0.118106	-0.03465	0.992396	0	-4.552	1.18915	0.338158	1	7.02906
20	1.74	-0.17816	-0.98398	-0.00704	0	0.976821	-0.176	-0.12185	0	0.118655	-0.02858	0.992524	0	-4.55259	1.18779	0.346877	1	6.99882
21	2.06	-0.17815	-0.984	-0.00312	0	0.976548	-0.17641	-0.12342	0	0.120899	-0.02504	0.992349	0	-4.55252	1.1879	0.350005	1	7.08984
22	2.18	-0.17742	-0.98412	-0.00626	0	0.976663	-0.17529	-0.12411	0	0.121044	-0.02814	0.992248	0	-4.55122	1.19245	0.343505	1	7.12967
23	2.22	-0.17686	-0.98419	-0.00914	0	0.976937	-0.17441	-0.12319	0	0.119644	-0.03072	0.992342	0	-4.55107	1.19387	0.348696	1	7.07630
24	2.3	-0.17689	-0.98422	-0.00347	0	0.976839	-0.17513	-0.12294	0	0.120395	-0.02513	0.992408	0	-4.55137	1.19195	0.344932	1	7.06196
25	2.34	-0.1779	-0.98403	0.005692	0	0.976421	-0.17724	-0.12325	0	0.122291	-0.01637	0.992359	0	-4.55147	1.19126	0.34926	1	7.07982
26	2.42	-0.17861	-0.98381	0.014984	0	0.976233	-0.17909	-0.12204	0	0.122748	-0.00717	0.992412	0	-4.55129	1.19269	0.349774	1	7.010
27	2.46	-0.17029	-0.98529	0.013933	0	0.977676	-0.17071	-0.12251	0	0.123089	-0.00724	0.992369	0	-4.55065	1.19582	0.337952	1	7.03784
28	2.5	-0.18304	-0.98308	0.006178	0	0.975707	-0.18243	-0.1213	0	0.120381	-0.01618	0.992596	0	-4.55139	1.19648	0.347067	1	6.96755
29	2.54	-0.16879	-0.98564	0.00423	0	0.978374	-0.16806	-0.12059	0	0.119566	-0.01622	0.992694	0	-4.55124	1.19664	0.349549	1	6.92604
30	2.58	-0.15244	-0.98831	0.003647	0	0.981009	-0.15176	-0.12079	0	0.119931	-0.01484	0.992671	0	-4.55064	1.19481	0.349391	1	6.93774
31	2.62	-0.14921	-0.98876	0.009368	0	0.98134	-0.14924	-0.12124	0	0.121281	-0.0089	0.992578	0	-4.55043	1.19298	0.347773	1	6.9642
32	2.66	-0.15037	-0.98849	0.016843	0	0.980995	-0.1513	-0.12148	0	0.122627	-0.00174	0.992451	0	-4.55048	1.19242	0.35015	1	6.97840
33	2.82	-0.14894	-0.98881	0.008759	0	0.98128	-0.14889	-0.12216	0	0.122094	-0.0096	0.992472	0	-4.54961	1.19656	0.35013	1	7.01687
34	2.94	-0.14964	-0.9887	0.008718	0	0.981311	-0.14959	-0.12104	0	0.12098	-0.00956	0.992609	0	-4.55009	1.19152	0.343693	1	6.95262
35	3.02	-0.15079	-0.98854	0.007585	0	0.981009	-0.15058	-0.12225	0	0.121995	-0.01099	0.99247	0	-4.55018	1.1897	0.349523	1	7.02240
36	3.1	-0.14924	-0.98879	-0.00424	0	0.981512	-0.14761	-0.12184	0	0.119845	-0.02235	0.992541	0	-4.5498	1.19206	0.349372	1	6.99819
37	3.14	-0.14893	-0.98878	-0.0116	0	0.981652	-0.14642	-0.12214	0	0.119076	-0.02957	0.992445	0	-4.54978	1.19213	0.35035	1	7.01632
38	3.18	-0.14841	-0.98882	-0.01416	0	0.981767	-0.1456	-0.1222	0	0.118773	-0.03204	0.992404	0	-4.5499	1.19071	0.349787	1	7.01985
39	3.22	-0.14896	-0.98877	-0.01218	0	0.981692	-0.14639	-0.12186	0	0.118708	-0.03011	0.992472	0	-4.55009	1.18775	0.33789	1	6.95262
40	3.26	-0.15012	-0.98865	-0.0055	0	0.981447	-0.14835	-0.12147	0	0.119273	-0.02363	0.99258	0	-4.55023	1.18689	0.346994	1	7.02240

Left4 Torso

Full Screen

Figure 7.10: XLS export example.

XML

With the well known and used XML output (see figure 7.11), the data can them be used as input for another program for further manipulation and processing.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<elems>
<elem name="Left4_Torso">
<matrix time="0.18">-0.00341114 -0.999994 -1.52527E-7 0.0 0.999994 -0.00341114
 ↪  7.23039E-5 0.0 -7.2304E-5 9.41126E-8 1.0 0.0 -4.56328 1.19874 0.383706
 ↪  1.0 -0.004142708 8.739153E-6 1.5258789E-5</matrix>
<matrix time="0.38">-0.00341107 -0.999994 -1.7801E-7 0.0 0.999994 -0.00341107
 ↪  1.09919E-4 0.0 -1.09919E-4 1.96932E-7 1.0 0.0 -4.56692 1.19875 0.37809
 ↪  1.0 -0.0062978948 1.0199222E-5 1.5258789E-5</matrix>
<matrix time="0.46">-0.00341107 -0.999994 -2.13309E-7 0.0 0.999994 -0.00341107
 ↪  8.19609E-5 0.0 -8.19612E-5 6.62669E-8 1.0 0.0 -4.56863 1.19876 0.372325
 ↪  1.0 -0.0046960134 1.2221705E-5 1.5258789E-5</matrix>
</elem>
</elems>
```

Figure 7.11: Example of the exported XML file.

7.2.3 Interface

After launching the application, a graphical interface appears and the user can select the server and/or agent log files to be manipulated. Then, a set of parse and process features are available for both the server and the agent logs.

The interface is composed of a top common banner area and bottom agent/server specific area. Figures 7.12 and 7.13 show, respectively, the available options from the agent and server perspectives. Supported in these two figures, it follows an explanation of both the graphical elements and the operation performed when they are selected.

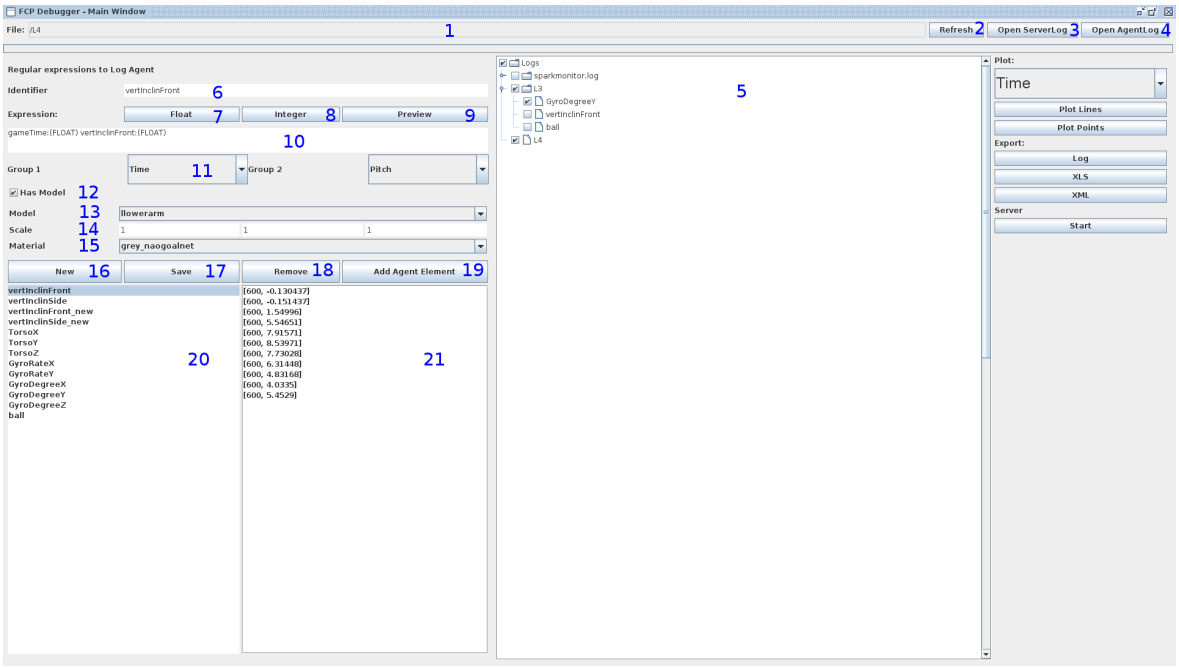


Figure 7.12: Log Analyser agent log selected

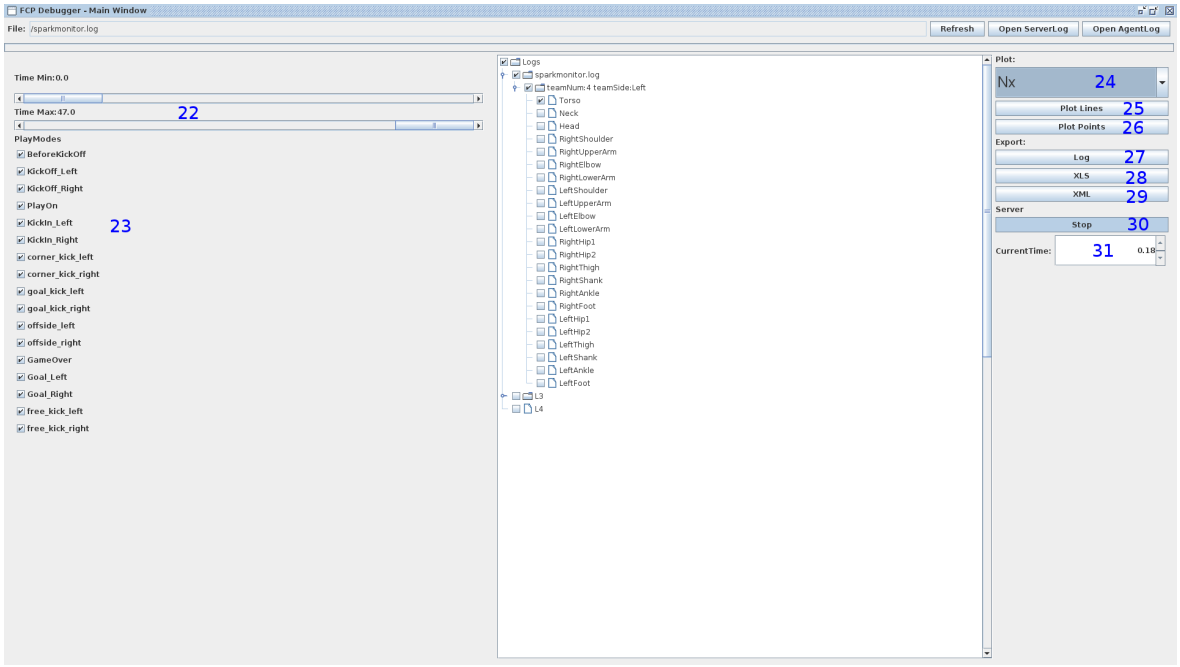


Figure 7.13: Log Analyser server log selected

1. File path of the current selected log.
2. Button used to refresh all the program. This is useful, for example, when the logs selected in the interface are changed because new simulation was run. This way, there is no need to select the log files again or restart the program. To refresh, the program begins by saving the current interface state (log tree selection, playmodes checkboxes, time, etc), then all the interface is reset and all the logs are parsed again. At the end, the initial interface state is applied and if any plot is visible, iti s refreshed too.
3. Let the user select a server log. This log, called ***sparkmonitor.log***, is generated by simspark if ***recordLogfile*** variable in ***simspark.rb*** file is set to ***true***. Only one server log can be used by the program at a time, corresponding to one simulation. After the file is selected, the program tries to parse it, and, if successful, presents in the log tree the agents present in the log, and its components (Ex: head, neck, etc).
4. Let the user select a agent log. The agent logs are the files generated from the standard output of the agents code. Multiple agent logs can be selected, as multiple agents can generate their log in one simulation. After the file is selected, the program parses it, and adds it to the log tree.
5. Tree representation of the logs loaded in the program and its components. Using this tree representation it is possible to select which components to be used in the multiple export options. In the server node, the children are the agents and the ball present in the server log. In the agent, its children represent regex expressions added by the user. First, the user creates a new regex expression with ***16***, then fill the options ***6,10,11-15***. Afterwards, the user needs to save ***17***. Finally, with a agent tree selected in the tree log, the button in ***19*** creates a new element based on the regex expression and adds it to the children of the agent log.
6. Identifier of the current selected regex expression (must be unique).
7. To simplify the regex visualization and its creation, there are two buttons corresponding to the

two most used expressions (Float, Integer), that are replaced with the correct regex when all the expression is applied. Click in this button to insert FLOAT macro in current caret position. It will be replaced with:
`[-+]?[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?`

8. Click in this button to insert INTEGER macro in current caret position. It will be replaced with:
`[-+]?[0-9]*`

9. Click to preview the effect of the current regex expression applied in the selected agent log.
10. Input box with the regex expression to use. Each parentheses pair represents a regex group, or in other words, a value to be retrieved when applying the regex expression to the log file. Each group has a meaning, it can represent the time, a transformation matrix element, pitch, roll, yaw or simply a custom value to analyse. For the former there are Val<n> from 1 to 4 in the combo box to be used. The groups combo boxes are automatically updated as the user changes the regex expression.
11. Groups combo Boxes actually present in the regex expression and its meanings.
12. Checkbox to indicate if the regex expression has a 3D model associated. If so, a group of configurations for the 3D model are presented, otherwise they are hidden. With this feature, when exporting in log format, a new visual element is created using the 3D model options in *13-15* and added to the server log. This way, when using a monitor to view the exported log, the selected model, a 3D model will appear in the positions extracted from the regex expression in *10*.
13. Select the 3D model to use. The ComboBox is filled with the known models used by the simulation server, plus any .obj files found in */usr/local/share/rcssserver3d/models* directory. The standard models StdUnit (Cylinder, Box and Sphere) are only displayed in the rcssmonitor3d, they don't appear in Roboviz.
14. Define the scale in X,Y and Z.
15. Defined the material to use in the model. The ComboBox is filled with the known materials used by the simulation server.
16. Create new regex expression. The new expression get as identifier the current date.
17. Save any changes to the selected expression.
18. Remove the selected expression.
19. Add selected regex expression to the selected agent log.
20. List with the identifiers of all the regex expressions the user has already created.
21. List where the first results of the preview appears.
22. Select the server time range to process.
23. Select simulation play modes to process.

24. Global group selection combo box. Used as a plot option.

25. Click to show Lines Plot. At least one log tree node must be selected. Then, the Group to be used in the plot can be selected globally in the combo box below the "Plot" label, or individually, in each log tree node. The former is used if the global group combo box has the value `"-"` selected.
    After clicking the button, a new window is created in which the plot is presented, using GRAphing Library (GRAL), modified a little. Each group has a different colour and when a plot point is clicked, a label appears containing the clicked point time and value. Furthermore, the GRAL [45] gives the option to zoom, move, print and export the plot as image with multiple possible formats using the right click menu (Wireless Application Protocol Bitmap Format (WBMP), Bitmap image file (BMP), Portable Network Graphics (PNG), Joint Photographic Experts Group (JPEG), Portable Document Format (PDF), Graphics Interchange Format (GIF), Scalable Vector Graphics (SVG), Encapsulated PostScript (EPS)). The images presented in 5.1 were generated using this export options.

26. Click to show Points Plot.

27. Click to export to a simspark log format. All the options in the interface are applied to the selected logs and its selected elements, then the server log is merged with the agent logs, resulting in the output log. This contains the 3D models information from the agent logs processing, so they can be visualized in a simulation monitor.

28. Click to export the selected logs/elements to XLS format. It makes uses of the Java Excel API (Only one node can be selected for export). [46]

29. Click to export the selected elements to XML format.(Only one element can be selected)

7.2.4 Configuration Save

As the program was being developed, it became useful to save some changes done in each program execution. For this purpose a XML file was created named *`configs.xml`* (see figure 7.14). In it, it is saved all the logs previously selected by the user, regex expressions created and its options. When the program is executed, the configuration file existence it is verified. If the file exists, all logs presented are loaded and all the regex expressions made available, otherwise the file is created.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configs>
  <expressions>
    <expression hasModel="false" identifier="vertInclinFront">
      <regex>gTime:(FLOAT) vertInclinFront:(FLOAT)</regex>
      <groups>
        <group>Time</group>
        <group>Pitch</group>
      </groups>
  </expression>
  <expression hasModel="true" identifier="ball">
    <regex>gTime:(FLOAT) .*BallPos (FLOAT) (FLOAT) (FLOAT)</regex>
    <scale x="1.0" y="1.0" z="1.0"/>
    <material>yellow</material>
    <model>StdUnitSphere</model>chr
    <groups>
      <group>Time</group>
      <group>Px</group>
      <group>Py</group>
      <group>Pz</group>
    </groups>
  </expression>
  </expressions>
  <openLogs>
    <log type="server">/home/test/fcp/sparkmonitor.log</log>
    <log type="agent">
      <fileName>/home/test/fcp/L3</fileName>
      <expressions>
        <expression identifier="vertInclinFront"/>
        <expression identifier="ball"/>
      </expressions>
    </log>
  </openLogs>
</configs>
```

Figure 7.14: Example of the configs.xml file.

7.2.5 Server Simulation

A great way to understand what is happening in a giving time is to see it appearing in the simulation monitor. This is the purpose of this feature, which simulates the rcssserver3d server. It bounds to the default rcssserver3d port(3200), waiting for monitors connections.

Starting from here, some options in the Log Analyser can be made visible in the monitors in real time.

It works by cloning the Ruby Scene Graph (RSG) message previous to the current selected time and applying all the Ruby Diff Scene (RDS) in between. After this aggregation, which creates the current time RSG message, the resulting message is sent to all the monitors connected to this server.

When a monitor connects to the server, the current time message is sent to it.

One must remember that the processed RSG message is the result of the options configured in the interface, especially the log tree selection, so it is possible to show in the monitors only some agents, or some agent elements or some processed regex expressions in the form of a 3D model. This make it easy to analyse only one agent leg movement in a full game, for example.

In the figure 7.15, we can see a possible setup to analyse the agent torso inclination. Only the Torso, Neck and Head nodes of one agent are selected and, because of that, only that elements appear on the monitor. There are also two plots opened with the Torso Nx and Ny of the agent 4 of the Left

team. When the user changes the time in *2* or click on a point in any of the graphs, all the elements are synchronized: the time in *2* will be updated; the plots opened will show a label with the value in the selected time; and the monitor will be updated too.

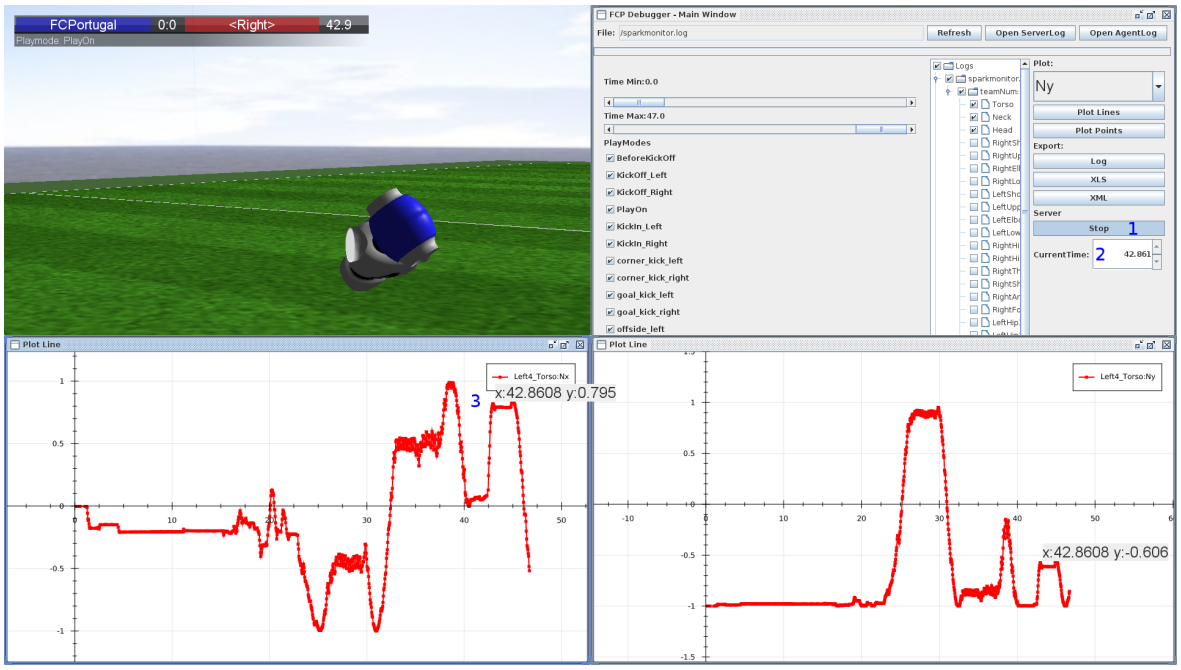


Figure 7.15: Server running with one plot selecting a time value where we see the agent falling with a pitch of approximately 45°.

Based on the figure showed above, the description of the server simulation is presented below:

1. Click to start/stop the server simulation. If it is stopped, the current time indicator will be invisible.

2. Change the server time using the arrows, introducing manually the value, or using the scroll wheel from the mouse. The possible time values are restricted between the server log last time, and its first time with all the agents connected.
   When the value changes, the current RSG message is sent to all the monitors connected, and all the plots visible show a label corresponding to the current time.
   The time will not change if zero elements are selected in the log tree.

3. Clicking in any plot point, will move the current time indicator to the plot point time, thus sending to the monitors connected to this server the new time RSG message. Furthermore, the other plots visible show a label corresponding to the current time.

7.3 XML Defined Behaviour Tester

There are different kinds of behaviours in the FC Portugal team, some of them are dynamically calculated in runtime (ex:Walk) and others are statically defined in XML files (ex:SlotBehavior, CPGBehavior, StepBehavior, ImprovedSlotBehavior, ImprovedCPGBehavior). For the last ones, the process of testing and improving was being done by changing the code to run tests for each behaviours,

compiling and then using a text editor for editing the XML files. For each change in the XML file, running a new agent executable was needed to visualize the result.
To improve the efficiency of this process, a new tool (see image 7.16) was created using GTK2, Scintilla and some agent code. GTK was used for the user interface and Scintilla is a open source library which provides text editing and XML syntax highlighting.

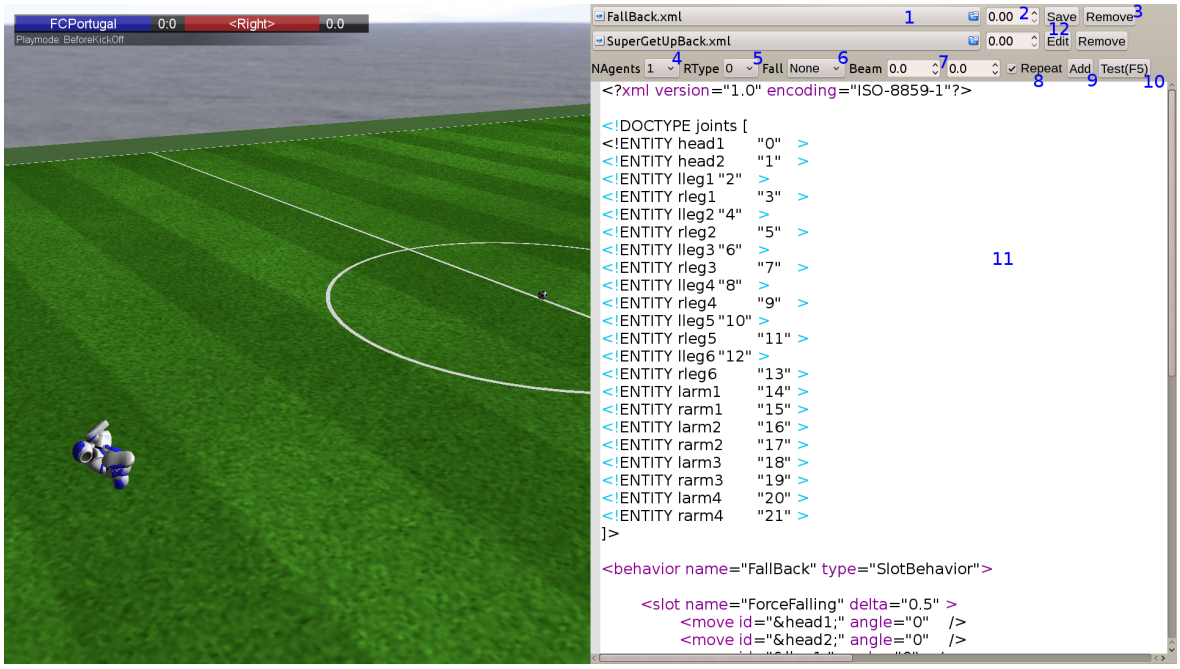


Figure 7.16: Behaviour Tester

Supported on figure 7.16, it follows an explanation of the features available in the application

1. Choose the behaviour file to use.
2. Edit time to wait after this behaviour ended.
3. Remove this line behaviour from the list.
4. Select how many agents to run in each test iteration.
5. Choose the robot type to use from the list of heterogeneous models.
6. Choose if the agent should fall back, front, side or not at all.
7. Choose first agent initial position
8. Select if the test iteration should be repeated.
9. Add new behaviour file.
10. Save file and run the test.
11. Editable area with syntax highlighting.
12. Save current editable file or choose to edit another file

Requisites: libgtk2.0-dev (Ubuntu), gtk2 (Arch Linux)

chapter 8

Conclusion

While objectives of this thesis were ambitious, they were mostly accomplished.

Starting with the optimizations, a good progress was done in terms of the process efficiency. With the optimization agent it is possible to start the process without changing any line of code. The separated agent also enables to have multiple agents optimizing in one simspark server, which is good for PCs with low resources or just for simplifying the number of consoles open and processes to control.

In terms of the behaviours, the newly optimized getup have good times, reducing by half the time taken, depending on the robot model. This is clearly an advantage in the game and can be used in conjunction with tactics so the best robots are used in specific situations.

The tools developed are quite handy. There is the Agent Monitor Window that can show variables in real time, for multiple agents. This in conjunction with sync mode, which can pause or change the game speed, is like a debugger, but with easier access, and with the possibility to send messages or values to callbacks in the code, so the agent can be changed in execution time, with the advantages that come with that.

The tester for XML Defined Behaviour is a utility that can load multiple behaviours, from different model types, in repetition and in execution time, reduces the time needed to rerun the agent for each change in the XML file and the time to open and edit different model types.

The last tool is the Log Analyser. It has lot of features, it can become useful for the developers to extract information from the logs, either the server one or the agents output. This information can then be processed and be redirected to different outputs, like plots, XML, XSL, monitor visualization, each one with its own possibilities.

In conclusion, this tools gives the team developers new ways to approach the problems, saving time and energy for other more important problems.

8.1 Future Work

In the FC Portugal project, as part of the RoboCup community, there is always work to be done and room for improvement. As such, some features that can be done in the future for the project or on top of this thesis are:

- The preconditions help reduce the time that some static behaviours consume when they are expected to fail. Other possibility would be to develop behaviours in order to make them adaptable, taking in account the known world state and sensors to produce the expected output, cycle by cycle. This way, in some cases, the agent could recover from unexpected behaviour variation.
- The optimization process could be further automated, with a flexible tool which should create the simulations, agents, run multiple types of optimizations and manage all the elements all by itself. Furthermore, the optimization re-sampling could be made parallel with one agent running each resample needed, further increasing the efficiency, which could help to optimize more variables and situations.
- The Log Analyser of the section 7.2 has already implemented the parser for the agents and server logs, and added some useful features. With this base, the tool has potential to grow and be extended based on the developers needs. The same applies to Agent Monitor Window and Behaviour Tester.
- The game is still a lot physical and the dispute for the ball is intense, so it could really help integrate some stabilization system using the walk and the arms.
- As new heterogeneous models are available for the competition, it could be interesting to see which types are better or have more potential at some specific behaviour, and incorporate this knowledge in the tactical roles of the game.
- For the code stability and to assess each commit, could be of use having a battery of tests with some criteria.
- For the students entering in the project, a wiki would be a good addition as it could help to explain what already has been done. In it, the overall architecture, major classes description, documentation, tools descriptions, future and current issues and some tips can be placed.

References

[1] *FC Portugal website*. [Online]. Available: http://paginas.fe.up.pt/~lpreis/FCPortugal.htm.

[2] *Robocup Objective*. [Online]. Available: http://www.robocup.org/about-robocup/objective/ (visited on 10/05/2014).

[3] *RoboCup Wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/RoboCup (visited on 10/03/2014).

[4] *Rescue Simulation League*. [Online]. Available: http://wiki.robocup.org/wiki/Rescue%5C_Simulation%5C_League.

[5] *Rescue League*. [Online]. Available: http://wiki.robocup.org/wiki/Robot%5C_League.

[6] *RoboCup@Home Official Website*. [Online]. Available: http://www.robocupathome.org/.

[7] *Nao Aldebaran Website*. [Online]. Available: http://www.aldebaran.com/en/humanoid-robot/nao-robot.

[8] *RoboViz Website*. [Online]. Available: https://sites.google.com/site/umroboviz/.

[9] *RoboViz Paper*. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-32060-6%5C_24.

[10] Wikipedia, *S-expression*. [Online]. Available: http://en.wikipedia.org/wiki/S-expression (visited on 10/02/2014).

[11] R. Community, *Simspark network protocol*. [Online]. Available: http://simspark.sourceforge.net/wiki/index.php/Network%5C_Protocol (visited on 10/03/2014).

[12] C. Bustamante, "Simspark Monitor Protocol", pp. 1–7, 2008. [Online]. Available: http://jeapres.ams.eng.osaka-u.ac.jp/~joschka/simspark/monitorprotocol.pdf.

[13] E. M. University, *Chapter 2- Robot Kinematics: Position Analysis*.

[14] M. Vukobratovic and B. Borovac, "ZERO-MOMENT POINT — THIRTY FIVE YEARS OF ITS LIFE", *International Journal of Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004. [Online]. Available: https://www.cs.cmu.edu/~cga/legs/vukobratovic.pdf.

[15] Ching-Long Shih, "Ascending and descending stairs for a biped robot", *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 29, no. 3, pp. 255–268, May 1999, ISSN: 10834427. DOI: 10.1109/3468.759271. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=759271.

[16] C. L. Shih and C. J. Chiou, "The motion control of a statically stable biped robot on an uneven floor.", *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 28, no. 2, pp. 244–9, Jan. 1998, ISSN:

1083-4419. DOI: 10.1109/3477.662765. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/18255942.

[17] Y. Zheng and J. Shen, "Gait synthesis for the SD-2 biped robot to climb sloping surface", *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 86–96, 1990, ISSN: 1042296X. DOI: 10.1109/70.88120. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=88120.

[18] S. Carpin and E. Pagello, "The challenge of motion planning for humanoid robots playing soccer", *Proceedings of the Workshop on Humanoid Soccer Robots of the 2006 IEEE-RAS International Conference on Humanoid Robots. 2006.*, 2006. [Online]. Available: http://www.researchgate.net/publication/228916772%5C_The%5C_challenge%5C_of%5C_motion%5C_planning%5C_for%5C_humanoid%5C_robots%5C_playing%5C_soccer/file/5046351b650452edfb.pdf.

[19] P. Sardain and G. Bessonnet, "Forces acting on a biped robot. Center of pressure-zero moment point", *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 34.5*, pp. 630–637, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=1325327.

[20] Q. Zhang, C. Dong, W. Xie, G. Sun, and J. Pan, "HfutEngine3D Soccer Simulation Team Description Paper 2014", Tech. Rep., 2014.

[21] V. Hugel and J. Nicolas, "L3M-SIM Team Description", Tech. Rep., 2014.

[22] A. S. Ashraf, A. A. Moosavian, F. G. Gohar, F. T. Shadpour, R. Moradi, and S. Moayeri, "Mithras3D Team Description Paper 2014", Tech. Rep., 2014.

[23] S. Abeyruwan, P. Nath, K. Poore, A. Seekircher, J. Stoecker, and U. Visser, "RoboCanes 3D Simulation League Team Description Paper 2014", Tech. Rep.

[24] H. Picado, M. Gestal, and N. Lau, "Automatic generation of biped walk behavior using genetic algorithms", *Bio-Inspired Systems*, 2009. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-02478-8%5C_101.

[25] J. Lima and J. Gonçalves, "Humanoid robot simulation with a joint trajectory optimized controller", *ETFA 2008*, 2008. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=4638514.

[26] L. Righetti and A. Ijspeert, "Programmable central pattern generators: an application to biped locomotion control", *Robotics and Automation, 2006. ICRA*, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=1641933.

[27] A. Ijspeert, "Central pattern generators for locomotion control in animals and robots: a review", *Neural Networks*, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608008000804.

[28] E. Kandel, J. Schwartz, and T. Jessell, *Principles of neural science*. 2000. [Online]. Available: http://www.just.edu.jo/FacultiesandDepartments/FacultyofEngineering/Departments/BiomedicalEngineering/Documents/Neuroscience%20Syllabus.pdf.

[29] S. Behnke, "Online trajectory generation for omnidirectional biped walking", *Robotics and Automation, 2006. ICRA 2006*, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=1641935.

[30] N. Shafii, S. Aslani, O. Nezami, and S. Shiry, "Evolution of biped walking using truncated fourier series and particle swarm optimization", *RoboCup 2009: Robot Soccer World Cup XIII. Springer Berlin Heidelberg*, pp. 344–354, 2010. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-11876-0%5C_30.

[31] N. Shafii, A. Abdolmaleki, R. Ferreira, N. Lau, and L. P. Reis, "Omnidirectional Walking and Active Balance for Soccer Humanoid Robot", *Progress in Artificial Intelligence*, pp. 283–294, 2013.

[32] S. Kajita, F. Kanehiro, and K. Kaneko, "The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation", *Robots and Systems*, 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=973365.

[33] S. Kajita and F. Kanehiro, "Biped walking pattern generation by using preview control of zero-moment point", *ICRA'03. IEEE*, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=1241826.

[34] J. Park and Y. Youm, "General ZMP preview control for bipedal walking", *Robotics and Automation, 2007 IEEE*, 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=4209488.

[35] R. Ferreira, L. Reis, A. Moreira, and N. Lau, "Development of an Omnidirectional Kick for a NAO Humanoid Robot", *Advances in Artificial Intelligence*, 2012. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-34654-5%5C_58.

[36] T. Sederberg, *Bézier Curves*. [Online]. Available: http://www.tsplines.com/resources/class%5C_notes/Bezier%5C_curves.pdf.

[37] J. Xuan, H. Jiang, and Z. Ren, "Pseudo Code of Genetic Algorithm and Multi-Start Strategy Based Simulated Annealing Algorithm for Large Scale Next Release Problem", *update*, 2011. [Online]. Available: http://chercheurs.lille.inria.fr/~xuan/page/project/nrp/tr2%5C_pseudocode.pdf.

[38] M. Dorigo, *Particle swarm optimization*, 2008. [Online]. Available: http://scholarpedia.org/article/Particle%5C_swarm%5C_optimization.

[39] N. Hansen, *The CMA Evolution Strategy*. [Online]. Available: https://www.lri.fr/~hansen/cmaesintro.html.

[40] H.-G. Beyer, "Covariance Matrix Adaptation Evolution Strategies CMA-ESs", [Online]. Available: http://scholarpedia.org/article/Evolution%5C_strategies%5C#Covariance%5C_Matrix%5C_Adaptation%5C_Evolution%5C_Strategies%5C_CMA-ESs.

[41] *CMA-ES Wikipedia*, 2014. [Online]. Available: https://en.wikipedia.org/wiki/CMA-ES.

[42] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer Science & Business Media, 2000, p. 33, ISBN: 1852332212. [Online]. Available: http://books.google.com/books?id=v9PLbcYd9aUC%5C&pgis=1.

[43] *Gtk2 Documentation*. [Online]. Available: https://developer.gnome.org/gtk2/stable/ (visited on 10/03/2014).

[44] Oracle, *Swing*. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/swing/ (visited on 10/02/2014).

[45] "GRAphing Library", [Online]. Available: http://trac.erichseifert.de/gral/.

[46] *Java Excel API Official Site*. [Online]. Available: http://jexcelapi.sourceforge.net.