FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Humanoid Robot NAO: Developing Behaviours for Soccer Humanoid Robots

Luís Miranda Cruz

Master in Informatics and Computing Engineering

Supervisor: Luis Paulo Reis (PhD.)

Supervisor: Armando Sousa (PhD.)

19^{th} July, 2011

Humanoid Robot NAO: Developing Behaviours for Soccer Humanoid Robots

Luís Miranda Cruz

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Pedro Abreu (PhD.)
External Examiner: Artur Pereira (PhD.)
Supervisor: Luis Paulo Reis (PhD.)

19^{th} July, 2011

Abstract

The topic of humanoid robot control has driven much research in areas such as robotics, artificial intelligence, computer vision, among others. Newer and better technologies have been developed, providing a wide range of humanoid robots and simulators, something that is very useful for this subject. Nowadays, humanoids are equipped with several degrees of freedom leading to very versatile movements but also to more complexity regarding the inherent problems.

Robocup is a robotic soccer international competition that promotes the research in these areas, including humanoid leagues, both on simulated and real environments. The FCPortugal team has already a good record for wins in RoboCup. The team intends to study these problems due to its participation in these humanoid leagues.

Therefore, this dissertation proposes the development of automated processes to optimize behaviours and generate sequences of behaviours for a given task regarding the environment. An optimization engine was extended and improved in order to provide an easy to use and generic optimization platform. Several kinds of kicks were developed and optimized using the Hill Climbing and Tabu Search methods, and good results were achieved with more powerful and faster behaviours. Also a planning approach was used to sequence different behaviours, providing a solution for avoiding obstacles and precisely control the position and orientation of the agent.

Resumo

O controlo de um robô humanóide é um tópico que tem impulsionado muita pesquisa em áreas como a robótica, inteligência artificial, visão por computador, entre outras. Mais e melhores tecnologias têm vindo a ser desenvolvidas estando disponível uma vasta gama de robôs humanóides e simuladores robóticos que ajudam a esta tarefa. Cada vez mais os humanóides são equipados com um elevado número de graus de liberdade, o que apresenta grande versatilidade de movimentos, mas acrescenta também a complexidade do controlo.

O RoboCup é uma competição internacional de futebol robótico que fomenta a pesquisa nestas áreas, integrando ligas de humanóides, tanto em ambiente simulado como real. A equipa FCPortugal apresenta já um bom historial de vitórias no RoboCup e dada a sua participação nestas ligas de humanóides, pretende estudar estes problemas.

Nesse âmbito, esta dissertação propõe o desenvolvimento de processos automáticos de optimização de movimentos e de geração de sequências de movimentos para realização de tarefas de robôs humanóides. Foi desrenvolvido um sistema genérico de optimização de modo a disponibilizar uma plataforma de optimização genérica e fácil de usar. Utilizando os métodos de optimização *Subir a Colina* e *Pesquisa Tabu*, foram desenvolvidos vários comportamentos que se revelaram robustos e rápidos. Para sequenciar de forma eficiente os diferentes comportamentos do robot humanóide, foi desenvolvida uma metodologia através da modelação do problema como um problema de planeamento, o que proporcionou um mecanismo para evitar obstáculos e posicionar o agente de forma precisa num desejado ponto do campo com uma determinada orientação.

"One day, in retrospect, the years of struggle will strike you as the most beautiful"

Sigmund Freud

Contents

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

AI	Artificial Intelligence
AP	Attention Point
API	Application Programming Interface
ASIMO	Advanced Step in Innovative MObility
BASIC	Beginners All-purpose Symbolic Instructional Code
CPG	Central Pattern Generator
DEI	Department of Informatics Engineering
DOF	Degree of Freedom
FEUP	Faculdade de Engenharia da Universidade do Porto
GA	Genetic Algorithm
HC	Hill Climbing
HCR	Hill Climbing Random
IP	Internet Protocol
LIACC	Artificial Intelligence and Computer Science Laboratory
MIEIC	Integrated Master in Informatics and Computing Engineering
PDDL	Planning Domain Description Language
SA	Simulated Annealing
SPL	Standard Platform League
TS	Tabu Search
TCP	Transmission Control Protocol
TFS	Truncated Fourier Series
XML	eXtensible Markup Language

Chapter 1

Introduction

This section describes briefly the motivation and goals behind this project, as well as its context and the document outline.

1.1 Context

This dissertation is related with humanoid robot agents designed to play soccer. The scientific field responsible for the design, construction, operation and application of robots is known as **Robotics** [oxf]. Robotics in antiquity and through the middle ages was used primarily for entertainment purposes. However, in 20th century, robotics suffered a boom with its first application in the industrial environment. In 1961, George Devol installed a robot into a General Motors factory, capable of lifting die-cut metal pieces and stacking them for the human workers [Hag03, SHE03, Cha11, rob].

Since then, robots have been pivotal to a business. Of course the industry is only one of the many application of robotics: it can be used on education, scientific research, entertainment, military, and so on.

A more specific kind of robots is **autonomous robots**. Autonomous robots are intelligent machines capable of performing tasks in the world without any form of external control for extended periods of time [Bek05]. This turns the robotics to be even more popular and to have even more applications.

Among the numerous initiatives related to autonomous robotics, appeared the robotic soccer. The idea of soccer playing robots was introduced in order to stimulate and challenge the research in many scientific areas [KAK+97].

In 1996, the first robotic soccer competition took place in Japan – the *preRoboCup-96*. Since then, hundreds of researchers throughout the world come yearly to engage themselves and their students in **RoboCup** [CFH+02]. RoboCup, originally named *Robot*

Soccer World Cup is an international competition where several researchers come together to prove the concepts they developed and present them applied in soccer matches. The best robot team wins the contest. There are different leagues, including leagues beyond the soccer domain, related to rescue or domestic applications.

Currently, RoboCup includes the following leagues:

- RoboCup Soccer
  - Humanoid
    - Teen Size
    - Kid Size
    - Adult Size
  - Middle Size
  - Simulation
    - 2D
    - 3D
  - Small Size
  - Standard Platform (SPL)
- RoboCup Rescue
  - Robot League
  - Simulation League
    - Rescue Agents
    - Virtual Robots
- RoboCup@Home
- RoboCupJunior
  - Soccer
  - Dance
  - Rescue

In one of the many leagues of RoboCup, is the humanoid league. In the humanoid league the game is played by teams of humanoid robots. Humanoid robots are robots with a human-like appearance designed to behave like humans. A very popular humanoid is NAO. NAO has several sensors and joints that make him physically capable of playing soccer. Further details about humanoid robots will be presented in 2.1.

Since 2000, a Portuguese team, known as FCPortugal[1], has been participating in Robocup competition. It is a joint project of the Universities of Porto and Aveiro and has already won several awards, including 3 World RoboCup International Competitions [RLPA08, RL01, RLO01]. FCPortugal has more than 60 international publications related to RoboCup, intelligent simulation and cooperative robotics [LR07, RLML10].

The team Research Interests include:

- Multi-Agent Systems (MAS) and Coordination in MAS;
- Intelligent Cooperative Robotics and Robotic Soccer (RoboCup);
- Intelligent Simulation, Agent Based Simulation;
- Soccer, Game Analysis, Strategic Reasoning and Tactical Modelling;
- Humanoid walking and skills development using optimization methodologies;
- Bridging the gap between simulated and real robotics.

1.2 Motivation

Robotics plays an important role at many levels of our society. Robots can be used in military operations, spacial missions, industrial tasks such as painting or assembling, education, art, among others. Usually, Robotics is applied for tasks that humans don't enjoy doing, some of them because they are dangerous, repetitive, dirty, or boring. Besides, they are useful due to their robustness, precision and/or autonomy. When well programmed, they can perform some tasks better than any human.

Recently, the humanoid robots have been becoming increasingly popular. Several research projects are being developed in order to make them dance, play musical instruments, perform household chores, and so on.

Humanoids can be a powerful way of human-machine interaction. It's natural that humans tend to interact freely with human-like entities. Historically, humans had constrained themselves to technology, for instance, with the monitor, keyboard and mouse. With humanoids, humans can experience new kinds of interactions, not too different from the usual ones on their social lives [WF].

Robocup as also integrated humanoids in some of its leagues. What makes RoboCup really interesting is the fact that in one competition researchers have to solve problems from different subjects related to Robotics, Artificial Intelligence, Computer Vision, Multi-Agent systems, Real-Time Reasoning, Machine-Learning, among others. Its popularity also turned out to be a way to attract public to these research subjects. This means that the

[1] Official website available at: http://www.ieeta.pt/robocup/

robot soccer is a really attractive application of technology. However, these technologies can be used for many other problems, creating a huge social and economic impact.

The soccer game requires humanoid players to be capable of walking in many directions, to kick the ball with high precision, to get up when necessary, or even to defend the ball, if we're talking about a goal keeper. Soccer is a fast game, which follows that motion plans have to strive for the generation of the fastest gaits [CP06]. The better the robot executes these behaviors, the higher is the team's performance. So, it is important for a team to have behaviours that are robust, efficient and adaptable to different situations.

Even with high quality behaviours, the humanoid has to autonomously choose the sequence of behaviours that fits the best a given task. This is a real challenge because it might have a large number of behaviours, and with that, there are even more possible combinations for the sequence, so it has to reason and choose on the fly the one that could give a faster and more robust conclusion of the desired task.

1.3 Objectives

The main goals for this dissertation are:

- Develop fast and efficient behaviours for humanoid robot that can be useful in a soccer game.
- Develop a generic optimizer that can be easily used for different kinds of behaviours.
- Develop an integrated method capable of planning the best sequence of behaviors for a given task in real-time.

In the scope of this dissertation some experimental methodologies will be developed in order to explore the results in different game situations with simulated games against well-known RoboCup teams. Besides, to validate the approach and prove the concept it's important to participate in robotic competitions, events, conferences and journals.

1.4 Document Outline

This document is organized in 5 chapters, of which the first one is this introduction.

The second chapter presents the Literature Review, portraying what is being done in this subject's area and different approaches and methodologies used to solve related problems.

The third chapter, Optimization of Behaviours, describes the optimizer with the implementation details, the experiments made as well as the achieved results.

The fourth chapter, Humanoid Behaviours Planning, presents a planning approach for the humanoid control. The developed architecture and experiments made are described and its results are presented.

Within the last chapter are exposed some conclusions about the work done and the achieved results. Also some thoughts about possible future work is presented.

Chapter 2

Literature Review

This chapter presents the literature review of this project providing background knowledge and some analysis of related work.

2.1 Humanoid Robots

An humanoid robot is a robot with a human-like appearance. It is a robot which structure has two legs, two arms and a head. Besides, its movement is expected to be human-like, using legged locomotion.

We are familiar with humanoids as they have been inspiration for many fictional stories. Examples of these are *C-3PO* and battle droids in "*Star Wars*" films, or even *NS-5* in "*I, Robot*". In addition, humanoids can be used as a research tool in several scientific areas. This section presents the most important humanoid robots as well as some related projects.

2.1.1 Honda ASIMO

In 2008, the Honda robot ASIMO conducted the Detroit Symphony Orchestra. It was a remarkable fact in the robotics community. The name ASIMO is an acronym for "Advanced Step in Innovative MObility". It can be seen in figure 2.1, it looks like a small astronaut, standing 130 centimetres and weighing 54 kilograms. The 34 DOF allows it to do things like walking with a tray, or handle a cart. It is capable of walking fast, up to 3km/h forward, change direction and walk up/down stairs smoothly. It can even reach 7km/h by adopting a special running gait [Hon07, SWA+02]. Although its popularity, ASIMO is still a really expensive technology [Hes02].

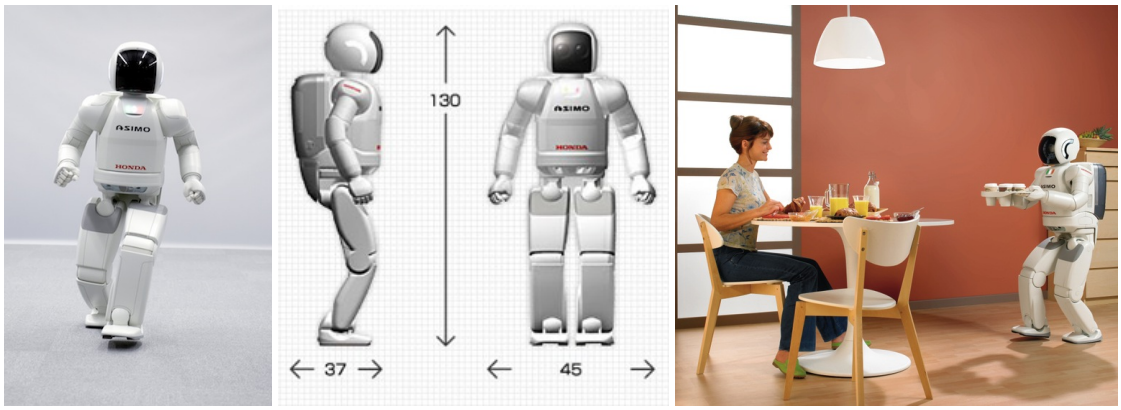


Figure 2.1: Honda ASIMO.

2.1.2 Sony QRIO

QRIO, which stands for "Quest for Curiosity", was an humanoid robot designed by Sony for use within the home. It could sing, dance and its successful development of integrated walking, jumping, and running. Its movement control technology lead to the world record of the first running humanoid robot [Gui04]. In one of its performances QRIO showed up conducting the Tokyo Philharmonic Orchestra in a rehearsal of Beethoven's Fifth Symphony. Nevertheless, in 2006, Sony announced the end of its Aibo robot line as well as the development of QRIO. QRIO stood at 58 cm tall and weighted 7.3Kg. It was able to interact with humans, thanks to its face and voice recognition systems, its speech synthesis and its remarkably fluid motion due to the equipped 38 DOFs [Gep04].

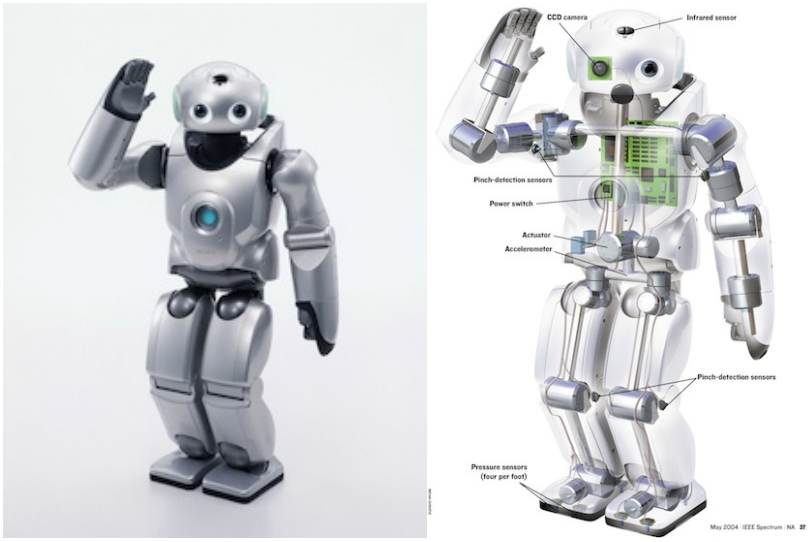
2.1.3 TOPIO - TOSY Ping Pong Playing Robot

TOPIO is a Vietnamese humanoid robot designed by TOSY Robotics to play ping-pong. It has 1.88m in height, 120kg in weight and 39 degrees of freedom. Ping Pong is a very challenging domain because it requires understanding of dynamic environments, accurate real-time vision, fast actuation, and intelligence to play the game with a winning strategy [Rai08]. It has an Artificial Intelligence module that allows to continuously improve itself while playing.

2.1.4 iCub

The iCub is an humanoid robot created for research in embodied cognition. It has the dimensions of a 3.5 year old child. It is distributed as Open Source under the GPL/FDL licenses both for software and hardware and the entire design is available for download at the project's official website [1]. This is very useful in research projects because

[1] Available at `http://www.robotcub.org`.



(a) Normal view of QRIO. (b) QRIO in detail. [Gep04]

Figure 2.2: Sony QRIO.

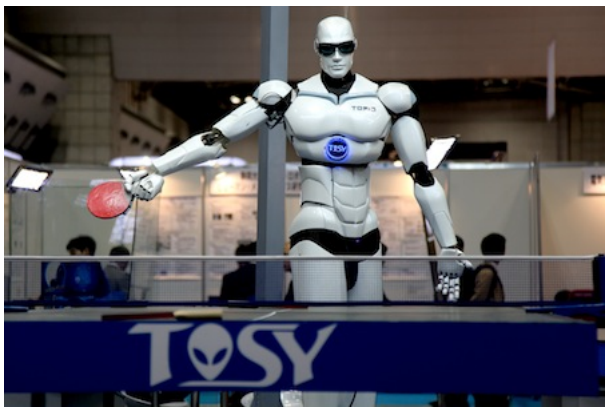


Figure 2.3: Tosyo ping-pong playing robot TOPIO.

users and developers can freely exploit, customize and benefit from the work of other users [MSV+08].

2.1.5 RoboNova Platform

RoboNova (cf. fig. 2.5) is a low cost humanoid developed by Hitec Robotics Company in Japan. Its original version comes with 19 DOF which can be controlled by programs created using RoboScript and RoboBasic IDEs. *RoboScript* is a graphical programming interface for beginners. For more advanced users, *RoboBasic* offers a very simple programming language based on BASIC specially designed for controlling humanoid robots [NKSD11].

Figure 2.4: iCub humanoid.

Due to its high motion capabilities, RoboNova has been widely used in many humanoid research projects, including in the development of autonomous dancing robots [GEKO09, CdSSMM08].

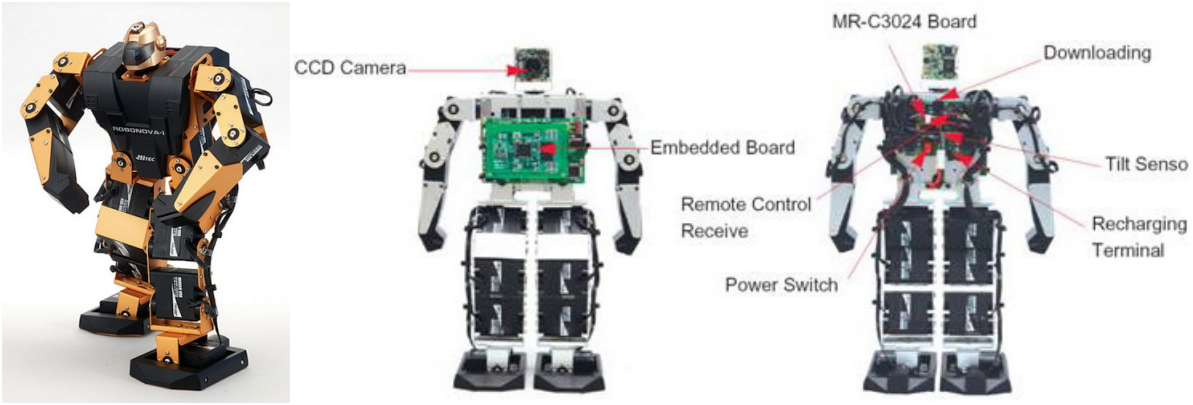


Figure 2.5: Robonova humanoid robot.

2.1.6 Aldebaran NAO

The humanoid robot NAO (cf. fig. 2.6) is built by the French company Aldebaran-Robotics[2]. It weighs 4.5 kg and stands 57 cm high. It was created with the goal of making available a performant biped robot to the maximum people. Humanoids like ASIMO and QRIO were either not available to researchers or only available to the few teams that have enough funding to support the cost and maintenance.

It is capable of recognizing features and human faces in the environment, to self-localize and to operate in the environment. Besides, this can be made while walking

[2]Official web site at: http://www.aldebaran-robotics.com/

which is a concern for the participants of RoboCup humanoid league. Another feature is the open architecture. Most of the embedded software, including the operating system can be changed by the user. Low level hardware access is also open to allow users to change joint control laws [GHB+08].

NAO has a total of 25 DOF. 11 DOF for the lower part, that includes legs and pelvis, and 14 for the upper part, that includes trunk, arms and head. Figure 2.7 presents the kinematics details.

Since 2008, NAO is the official robot for the RoboCup Standard Platform League.

Figure 2.6: The NAO developed by Aldebaran Robotics, now used in the RoboCup SPL.

2.2 Humanoid Simulators

In general, humanoid robots are a really expensive technology, as well as its maintenance. The use of simulation environments provides many advantages for research, development and test. For instance, in real environments every time a simple change is made in the agent, it's needed to reinitialize the robot and reinstall the new version all over again. Besides, all tests can be done without damaging the real robot and it's easier to retrieve detailed information about the execution.

The following sections present some of the most popular simulators.

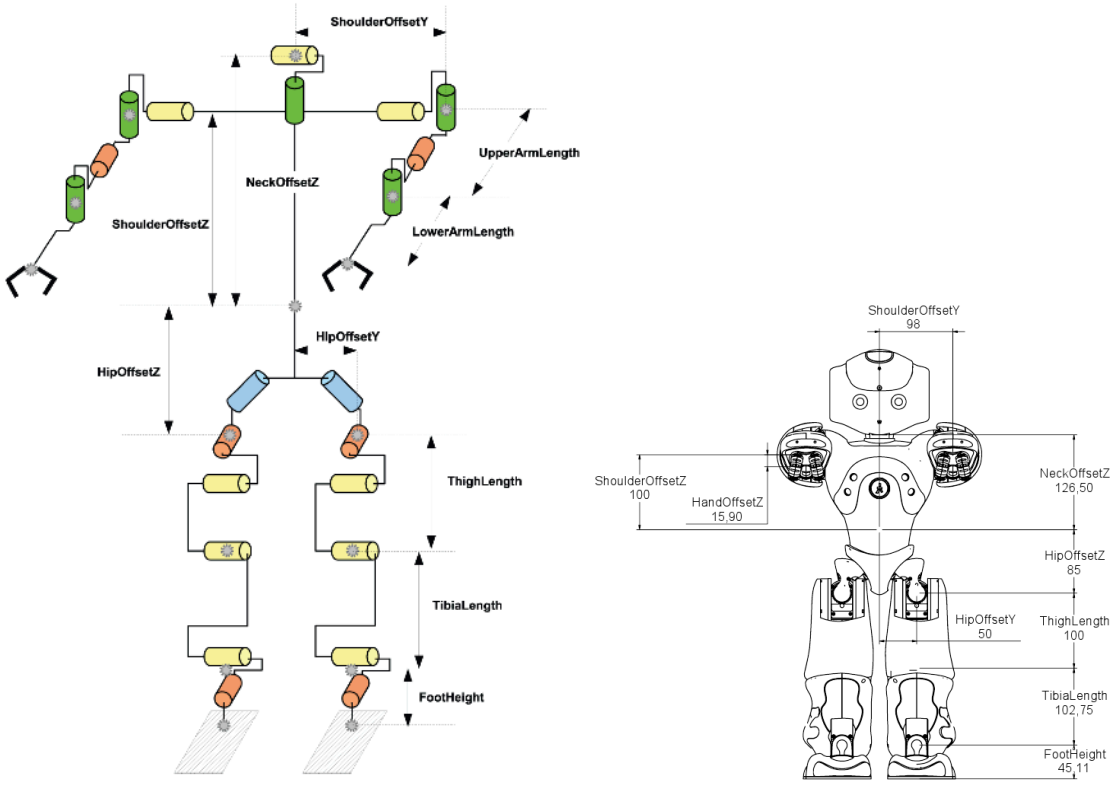
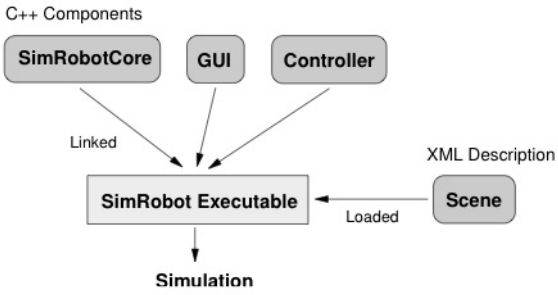


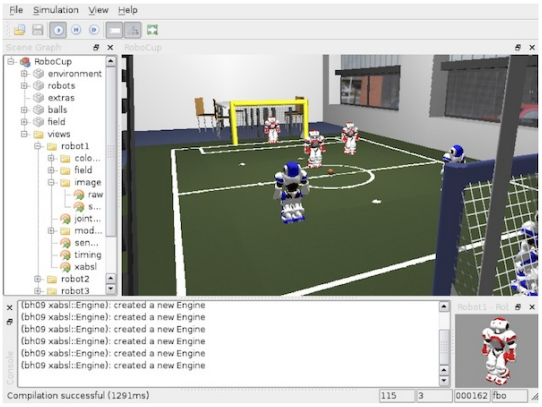
Figure 2.7: Detailed kinematics of NAO.

2.2.1 SimRobot

SimRobot [LSR06] is a simulator developed in the context of RoboCup by the B-Human team[3] [RLB+08, RLM+]. Unlike most of simulators, which have a client/server architecture, SimRobot consists of several components linked in a single application, as shown in figure 2.8a. This approach has been chosen to offer a more comprehensive debugging, allowing to halt or stepwise the simulation without any concurrences.



(a) The modules of SimRobot and their dependencies. Source: SimRobot – Development and Applications [LR08].



(b) SimRobot with the tree view on the left and two scene views on the right. [RLM+09]

Figure 2.8: SimRobot simulator.

The simulator has the following components: the *SimRobot core*, the *simulation scene*, the *user interface* and the *controller*.

The *SimRobot Core* models the robots and the environment, simulates sensor readings and executes commands given by the user or the controller. The *simulation scene* is a description of the robots and the environment, modeled using a XML specification language called RoSiML[4] [GZLR+05]. It consists of an external file that can be loaded at runtime.

The *user interface* (cf fig. 2.8b) provides the graphical visualization and user interaction of the simulation. It allows the user to move objects around the scene, so that different situations can be easily modelled, gives a control for direct manipulation over the actuators and has a graphical visualization for sensors.

The *controller* is the component that implements the sense-think-act cycle. In each simulation step, it reads the sensors, plans the next action, and sets the actuators to the desired states. This controller has to be provided by the user, it contains the control software of the robots which usually is identical to the software running in the real robots [LR08].

[3] B-Human's Official Website available at http://www.b-human.de

[4] Robot Simulation Markup Language. XML Schema is available at: http://www.informatik.uni-bremen.de/spprobocup/RoSiML.html

2.2.2 Webots

Webots™ is a robotic simulation environment used to model, program and simulate mobile robots [Mic98]. A large set of sensors and actuator is available to equip each robot and an integrated IDE is provided to program the controllers. Once tested in simulation, the controllers can be transferred to real robots, including the humanoid robot NAO. This is, both real and simulated robots can be programmed using the same API, making the source code of a robot controller compatible between the simulator and the real robot. Figure 2.9 depicts the Webots™'s user interface.

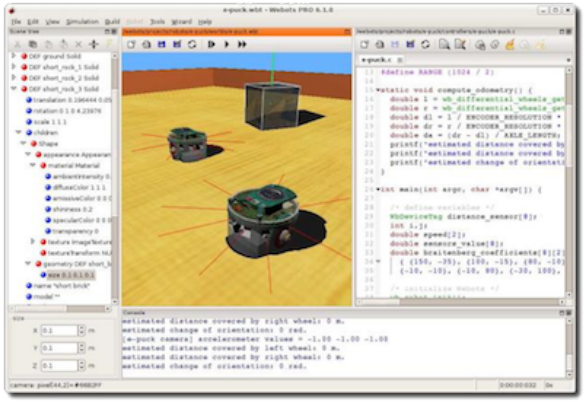


Figure 2.9: Webots simulating e-puck robot with a tree-view of the objects on the left and a source code editor on the right.

2.2.3 OpenHRP3

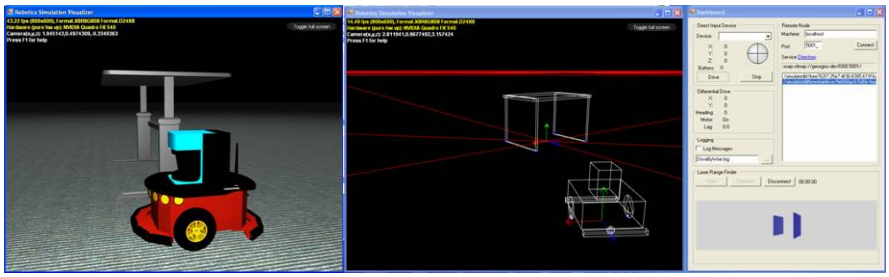
OpenHRP3 (Open Architecture Human-centered Robotics Platform version 3) is a simulator and motion control library for humanoid robots developed at the National Institute of Advanced Industrial Science and Technology, Kanehiro et al. (2004). It's a free tool that can simulate the direct dynamics of open and closed chains of multi-body systems. It includes a library of position, force/torque, vision, and inclination sensors, which are essential for developing control systems for the robots. The software also includes simple controllers for locomotion, balancing, a walking pattern generator as well as collision detection and avoidance schemes. Advanced controllers can be developed by the users. Besides, standard tools for plotting simulation results, OpenHRP has a 3D visualization interface to display the simulation.

The software developed in OpenHRP can be directly used in the HRP humanoid robots [KHK+08] but this is not possible for other humanoid robots. The main reason is that several features in OpenHRP cannot be modified by the user. For example, the actuator models are fixed, and compliance in the robot joints or the robot links cannot be added

to the model. Furthermore, there is very little documentation available for OpenHRP and since this is a free tool there is also no support from the developers [MCDB+10].

2.2.4 Microsoft Robotics Studio Simulator

Microsoft Robotics Studio (MSRS) is a Windows-based environment for robot control and simulation. It provides a set of useful tools, including a three-dimensional simulator. In the simulator, entities can be created in order to include robots and objects like walls, obstacles, floors, etc. Each entity is provided with a set of callback functions for dealing with events such as colisions and frame updates. It integrates the physics simulator PhysX created by Ageia. Entities in the simulation environment specify a physical description with parameters such as friction coefficient, mass and center of gravity. Screen shots of the simulator can be seen in figure 2.10 [Jac07].



(a) Modular robot base with differential drive, laser range finder and bumper array.

(b) Physics primitive view.

Figure 2.10: Screen shots of Microsoft Robotics Studio Simulator.

2.2.5 SimSpark

SimSpark is a multi-agent simulation system and is used as the official simulator for the RoboCup Simulation League competition [BA08]. Although it was created for robotic soccer simulation, the simulator core is the most generic possible and all soccer specific details are contained in a set of plugins [BDR+10]. The simulator is designed in a client/server architecture having three main kinds of components: agent, server and monitor (cf fig. 2.11).

An agent is the software that drives the behaviour of a robot. Usually, they communicate with the simulation server via TCP sockets.

The monitor is the component responsible for the graphic visualization of the simulation, and can be optionally executed. It connects to server and continuously receives a data stream with the information needed to render the scene. SimSpark provides different robot models for use by agents. The RoboCup 3D Soccer Simulation League currently

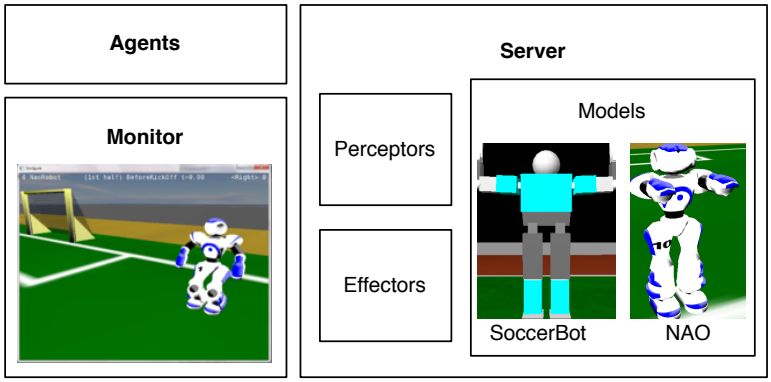


Figure 2.11: General architecture of SimSpark architecture.

uses a model of NAO humanoid. As shown in figure 2.12, the model is quite similar to the real NAO [BDR+10].

(a) real robot (b) virtual robot

Figure 2.12: The humanoid robot NAO: real vs simulated. Adapted from [BDR+10].

The server is the system core that hosts and manages the simulation process. It is responsible to constantly update the simulation state. Every interactions between the objects, collisions, gravity, and so on, are modelled by the server. Another responsibility of the server is to keep track of all the connected agents. The server provides perceptors to report the information about the senses of an agent, and provides effectors which allow agents to perform actions within the simulation.

The main components of the server are the simulation engine, the object memory management and the physics engine, as illustrated in the figure 2.13. The simulation engine binds all components together, advances simulation time in a main run loop, and calls various plugins to handle the communication with agents and monitors [BA08]. The object memory management is handled by an application framework called *Zeitgeist* [FJL+88].

Rigid-body physics calculations are handled by an external library, called *Open Dynamics Engine* (ODE) [S+05] which is integrated into the overall architecture of the system.

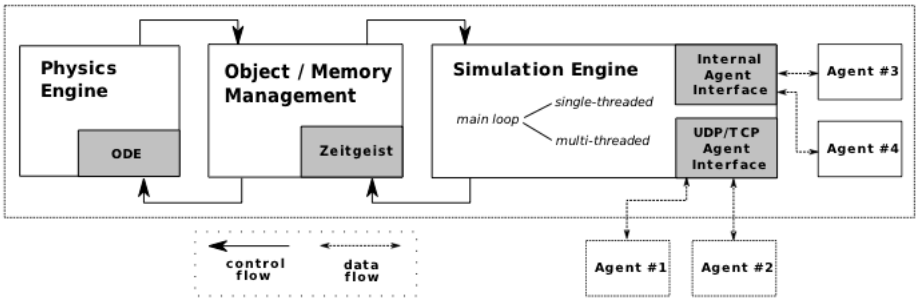


Figure 2.13: Detailed architecture of the simulation core. Adapted from [BA08].

2.3 Humanoid Behaviours

In this context, an humanoid behaviour is a set of trajectories of robotic joints composed in order express the control of the robot. Once specified it can be executed as many times as necessary. Behaviours can be expressed at the motor, skill or task level.

At motor level, behaviours are expressed by prescribing commands directly to system's actuators. At skill level, behaviours express the capabilities of the agent, for instance, walking, getting up, and so on. Skills are models expressed as parametrized programs for motor level controllers, without the ability to strategize about objectives or encode semantics about the world. Finally, task level behaviours are skills performed in order to achieve specified goals [JM03].

Some related research, including the work done by FCPortugal team, is presented in the following sections.

2.3.1 Skill Level Behaviours Generation

Defining a skill by composing several commands at motor level controllers commands can be a very difficult task. In order to make things easier, a skill can be defined with the trajectory for each involved joint. These trajectories can be generated based on a sequence of start and end points in the joint space during some amount of time [Pic08, PGL+09].

Some of the methods used by FCPortugal to define skill level behaviours are described in this section.

2.3.1.1 Step-based Behaviours

The step-based method was the first method used by FCPortugal. It generates behaviours using step functions. A step function (or staircase function) is a discontinuous function consisting of a series of constant functions, each one defined in some interval of time [Pic08].

So a joint trajectory will be defined as a step function in which the amplitude is the desired angle on each interval of motion. The function is described as follows:

$$f(t) = \sum_{i=0}^{n} \theta_i \cdot \mathbf{1}_{A_i}(t), \forall t \in \mathbb{R} \tag{2.1}$$

where n is the number of intervals, A_i is the interval i, and θ_i is the desired angle for joint at interval i. $\mathbf{1}_{A_i}(t)$ is the indicator function of A_i and is defined as follows:

$$\mathbf{1}_{A_i}(t) = \begin{cases} 1, \text{ if } t \in A_i \\ 0, \text{ if } t \notin A_i \end{cases} \tag{2.2}$$

The main advantages of this method are:

- Simple to understand;
- Simple to implement;
- Simple to define target trajectories (target angles and tolerances).

The main drawbacks of step-based method:

- Time from current angle to target angle is unpredictable;
- No control over the angular velocity trajectory;
- The same gain is used for all joints;
- Sensitive to overshoot reactions at the control level;
- The syntax of the configuration file is not user-friendly;
- The model is not flexible.

In FCPortugal team, a step-based behaviour is specified in a XML file, consisting of a sequence of steps, and each of them has a sequence of joint positions, corresponding to the target angles that might be achieved in the step. One step finishes its execution when all the joints are approximately at the specified angle.

2.3.1.2 Slot Based Behaviours

Another approach to generate behaviours used by FCPortugal is the slot-based method. Slots are intervals of time where several joints are moved in parallel. This allows to define not only the trajectory points, but also the time in which those angles should be achieved.

Comparing with step-based method, this approach allows to have control over the transition time of the current point to the target point. So, instead of defining the states at each interval of time, in slot-based methods transitions are specified.

In order to have smooth transitions between the current point and target point, sine interpolation is used. This works as follows: in each slot, the controller will interpolate between the current point and next point by performing a sine-like trajectory generated by the following expression:

$$f(t) = A\sin(\frac{\phi_f - \phi_i}{\delta}t + \phi_i) + \alpha, \forall t \in [0, \delta] \tag{2.3}$$

where $f(t)$ is the trajectory function, δ is the duration of slot in milliseconds ϕ_i is the initial phase (which will influence the initial angular velocity), ϕ_f is the final phase (which will influence the initial angular velocity), A is the amplitude and α is the offset. A and α are calculated using the following expressions:

$$A = \frac{\theta_f - \theta_i}{\sin(\phi_f) - \sin(\phi_i)} \tag{2.4}$$

$$\alpha = \theta_i - A \cdot \sin(\phi_i) \tag{2.5}$$

where θ_i and θ_f are the initial and final angles, respectively, and should be defined between $-\pi$ and π.

In the FCPortugal agent, this behaviours are described in XML files. In each behaviour is described a sequence of slots. Each slot has a time parameter and is composed by a series of moves for the involved joints, which will be executed at the same time. The table 2.1 describes these classes.

An example of a behaviour description can be seen in listing 2.1. Several slots are defined and each of them have a set of joint moves.

Listing 2.1: Extract from a slot behaviour XML specification (KickLeft Behaviour of the FCPA-gent) - February 2011.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <behavior name="KickRight" type="SlotBehavior">
3     <slot delta="0.170" />
4     <slot name="shiftWeight" delta="0.400">
```

Slot Behavior	
Class	**Properties**
Slot	name delta(δ)
Joint	joint identifier target angle (θ) kp, ki, kd

Table 2.1: Description of the classes in a Slot Behaviour specification.

```
        <move id="&lleg2;" angle="-11" />
        <move id="&lleg6;" angle="11" />
        <move id="&rleg2;" angle="-11" />
        <move id="&rleg6;" angle="11" />
        <move id="&lleg3;" angle="8" />
        <move id="&rleg3;" angle="8" />
        <move id="&lleg4;" angle="-14.5" />
        <move id="&rleg4;" angle="-14.5" />
        <move id="&lleg5;" angle="12.5" />
        <move id="&rleg5;" angle="12.5" />
    </slot>
    <slot name="kickLegUp" delta="0.500">
        <move id="&rleg3;" angle="20" />
        <move id="&rleg4;" angle="-40" />
        <move id="&rleg5;" angle="20" />
        <move id="&lleg3;" angle="0" />
        <move id="&lleg4;" angle="0" />
        <move id="&lleg5;" angle="0" />
    </slot>
</behavior>
```

The main advantages of this method are:

- Simple to understand and implement;
- Time from current angle to target angle is controlled;
- Some control over the angular velocities trajectories;
- The model is flexible;
- The motion description language is user-friendly and well structured.

The main drawbacks are:

- It is not possible to define more complex sinusoidal shapes;
- The angular velocities trajectories are not completely controlled.

CPG Behaviour Parameters	
Class	**Properties**
Behavior	delta(δ)
Pattern	joint identifier target angle (θ) period phase amplitude

Table 2.2: Description of the classes in a CPG Behaviour specification.

2.3.1.3 Central Pattern Generator

Central pattern generators are biological neural networks that can produce coordinated multidimensional rhythmic signals, under the control of simple input signals [RI06, Pin07]. They are found both in vertebrate and invertebrate animals for the control of locomotion. The concept of neural networks can also be applied in robot control in order to generate a suitable pattern for a walking motion.

The input parameters are described by the vector $a = (a_r, a_p, a_y)$ which corresponds to the lateral swing amplitude, forward swing amplitude and rotational amplitude, respectively. A gait phase, ϕ_{gait}, varies between $-\pi$ and π. ϕ_{leg} represents the phase of a leg and will correspond to $\phi_{gait} - \frac{\pi}{2}$ for the left leg and $\phi_{gait} + \frac{\pi}{2}$. The step of one leg during the whole gait is divided into five complex stages: *Shifting*, *Shortening*, *Swinging*, *Loading* and *Balance* [Rei10, RRL11]. Further details can be found in [Pic08].

The main advantages of CPG behaviours are:

- More complex shapes are possible;
- Angular velocities trajectories are completely controlled;
- Allows several gaits with just one generator (forward walk, backward walk, sided walk, curved walk, curved walk and turn on the spot);
- The generated gait is very similar to the natural human behaviour.

The main drawbacks are:

- Requires complex knowledge about human behaviours
- The shifting stage leads to slower movements.

A behaviour specification file for a CPG behaviour consists of a series of patterns for each specified joint. The table 2.2 describes the parameters used to specify the behaviour.

2.3.2 Task-level Behaviours

In task-level behaviours it's intended to combine the known behaviours in a sequence ensuring that the task is successfully completed. Several researches are being made on this subject.

An interesting approach was adopted by Ogawara [OTI+00], in which the tasks are learned by watching an human hand performing them. The idea is to represent actions in symbolic task models, acquired through observation. For each task, in the acquisition process is specified a set of *attention points* (APs), dividing the task in the steps that requires attention and analysis. Through observation, the system creates a model of the task classifying the actions between the APs, taking account of attributes like in which hand is the performing, the absolute position in 3D space, the kind of action (if it's grasp, release, pour, hand over), and many others.

Another interesting feature in this approach is the assignment of priorities to the attributes. For instance, the information of which was the hand that performed the task is expected to be less important than the kind of action. First, the robot tries to perform the task exactly as the described in the model. If it fails, then it will iteratively ignore the attributes with lower priority. This task model is flexible enough to apply an acquired task in different environments [OTI+00].

2.4 Optimization

As long as humankind exists, we strive for perfection in many areas. We want to reach a maximum degree of happiness with the least amount of effort. In our economy, profit and sales must be maximized and costs should be as low as possible [Wei09].

Therefore, as usually, there is a science to deal with this problems - Optimization. It refers to find the best elements from a set of possible alternatives, according to a criteria.

Thus, an optimization can be stated by a set of parameters, known as **decision parameters**, describing the alternatives, and a mathematical function, known as **objective function**, that analyses each alternative according to these parameters. The objective function gives a quantity to classify a candidate solution of the problem. If the goal is to maximize the function, it is called **utility function**, otherwise, if it's to minimize, then it's called **cost function**.

There are many approaches on solving optimization problems. In the following sections, a classification of optimization solving methods is presented, as well as the most popular algorithms.

2.4.1 Classification

2.4.1.1 Deterministic and Stochastic Algorithms

Optimization algorithms can be divided in two basic classes: deterministic and stochastic algorithms. Deterministic algorithms are used when the utility of a possible solution can be efficiently calculated. In each step of a deterministic algorithm exists at most one way to proceed.

On the other hand, stochastic algorithms contain instructions that use random numbers in order to decide what to do or how to modify data. Such methods apply in the usual case where a closed-form solution to the optimization problem of interest is not available and where the input-information into the optimization methods may be contaminated with noise [Wei09, WW04].

2.4.1.2 Individual and Population-based Algorithms

Another kind of classification that can be made it's individual or population-based methods. Individual-based methods deals only with one possible solution in each iteration. Conversely, in population-based methods multiple alternatives are handled at the same time. [Pic08]

2.4.1.3 On-line and Off-line Optimization

According with the use case, methods can be classified with regard to the required speed. It's possible to distinguish **on-line optimization** in which problems need to be solved quickly in a time span between ten milliseconds to a few minutes. In order to find a solution in this short time, optimality is normally traded in for speed gains. With **off-line optimization**, time constraints are less important and the user can wait even days if that means obtaining an optimal or close-to-optimal solution [Wei09].

2.4.2 Hill Climbing

The Hill Climbing (HC) algorithm is simple to implement and performs well in several situations, achieving reasonable results. It starts with a arbitrary guess of the solution, and then attempts to find a better solution by analysing the neighbour alternatives, which are obtained by making minor changes to the guess. It terminates when reaches a state where no neighbour has a better value.

Hill-climbing algorithms typically choose randomly among the set of best successors if there is more than one. It often makes rapid progress towards the solution because it is usually easy to improve from a bad state. Unfortunately, hill-climbing is not so good when [RN10]:

- There is a local maximum, which is a peak that is higher than each of its neighboring states but lower than the global maximum.
- There is a plateau, which is a flat area of the state-space landscape. It can be a flat local maximum, from which no better state, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau.

Hill climbing is good for finding a local optimum, but it is not guaranteed to find the best possible solution (the global optimum) out of all alternatives. The algorithm is shown in algorithm 1.

Algorithm 1 Hill Climbing search algorithm.

```
1: current ← GetInitialSolution()
2: loop
3:     neighbour ← a highest-valued successor of current
4:     if neighbour.VALUE < current.VALUE then
5:         return current.STATE
6:     end if
7:     current ← neighbour
8: end loop
```

2.4.3 Simulated Annealing

Simulated Annealing (SA), also known as *Monte Carlo annealing* and *statistical cooling*, is a stochastic method to avoid getting stuck in local, non-global minima, when searching for global minima [DA89]. It was independently proposed by Kirkpatrick [KGV83] and Černỳ [Čer85], and further studied in detail by Aarst, Laarhoven and Korst [AVL85, AK88, VAL94].

Annealing, in metallurgy, is the process used to temper or harden metals and glass. This procedure hits the material to a high temperature so that the material is liquid and the atoms can move freely. The temperature of the material is then slowly decreased so that, at each temperature, the atoms can move enough to begin adopting the most stable orientation. If the material is cooled slowly enough, the atoms are able to achieve the most stable orientation.

The algorithm is presented in 2. When it's time to choose the next state, instead of choosing the best state, however, it picks a random state. If the random state has a better value, it is accepted. Otherwise, it accepts the move with some probability less than 1. The probability decreases as the state becomes worse, and also as the "temperature" T goes down. In the beginning of execution the temperature is high, and decreases over time. So worse states are more likely to be accepted at the start, but more unlikely as T decreases.

Algorithm 2 Simulated Annealing.

```
1:  current ← GetInitialSolution()
2:  for t = 1 to ∞ do
3:      T ← schedule(t)
4:      if T=0 then
5:          return current
6:      end if
7:      next ← a randomly selected successor of current
8:      ΔE ← next.VALUE − current.VALUE
9:      if ΔE > 0 then
10:         current ← next
11:     else
12:         current ← next only with probability e^(ΔE/T)
13:     end if
14: end for
```

2.4.4 Tabu Search

Tabu Search (TS) is a variant of HC method, proposed by Glover [Glo86, G+89, Glo90] in 1986. This algorithm maintains a list of the k last visited states in order to prevent a previous state from being repeated. With this, it's preferable to visit worse states rather than repeating them. As well as improving efficiency by avoiding cycles, this improvement can allow to escape from a local optima, since it can accept worse states [GHP+97, RN10]. The pseudo-code of this method is presented in algorithm 3.

Algorithm 3 Tabu Search algorithm.

```
1:  current ← GetInitialSolution()
2:  evaluation1 ← Evaluate(current)
3:  tabuList ← {}
4:  while TerminationConditionsNotMet() do
5:      neighbours ← GetNeighbourhood(current)
6:      for i = 1 to Length(neighbours) do
7:          if NotMember(neighbours[i], TabuList) then
8:              Add(TabuList, neighbours[i])
9:              evaluation2 ← Evaluate(neighbours[i])
10:             if evaluation2 < evaluation1 then
11:                 current ← neighbours[i]
12:                 evaluation1 ← evaluation2
13:             end if
14:         end if
15:     end for
16: end while
17: return current
```

2.4.5 Genetic Algorithms

The Genetic Algorithm (GA) was developed by Holland [Hol75] and it is an optimization method bases on the principles of genetics and natural selection. A GA allows a population composed of many individuals to evolve under certain selection rules to a state that maximizes the "fitness" (i.e., minimizes the cost function) [HH04]. In this case, an **individual** (also called **chromosome**) is a set of parameters, known as **genes**, and represents a possible solution to the optimization problem.

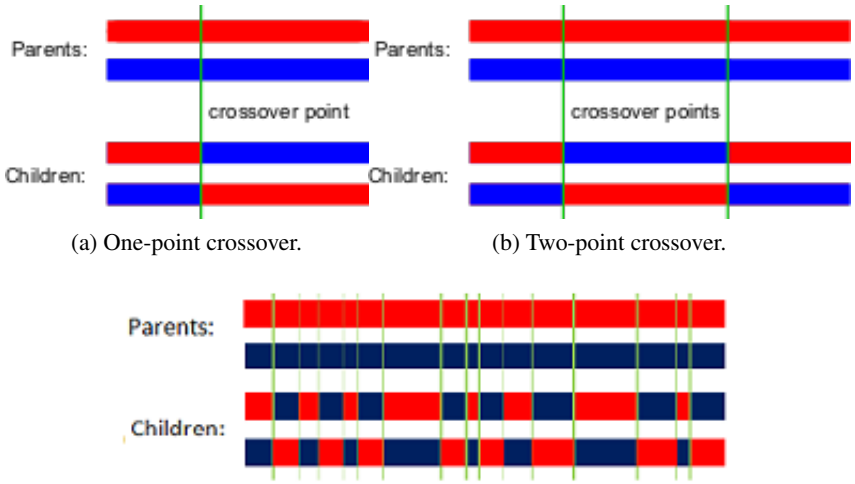
The algorithm starts by creating a new population of individuals. Then, it starts the evolution by applying several operators to the population, generating a new one. Along the algorithm, a sequence of new populations is generated.

The simplest form of GA, involves three types of operators: *selection*, *crossover* and *mutation*.

- **Selection** This operator selects chromosomes in the population for reproduction, accordingly to a given probability. The fitter the chromosome, the more times it is likely to be selected to reproduce.
- **Crossover** This operation randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100. The crossover operator roughly mimics biological recombination between two single-chromosome organisms.
- **Mutation** This operator randomly flips some of the genes in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (eg. 0.001) [Mit98].

There are different kinds of *crossover*: **one-point crossover**, **two-point crossover** and **uniform crossover**, among others. The simplest one, one-point crossover, is performed by selecting a single locus for crossover. All data beyond that point in both individuals is swapped between the two parent individuals. The two-point crossover is a variation of the previous, but instead of choosing a single point, two points are chosen. The last one, uniform crossover, uses a fixed probability to mix each gene of the parent individuals. Unlike, one and two-point crossover, the Uniform Crossover enables the parent chromosomes to be mixing genes rather than segments of genes. These methods are illustrated at figure 2.14

In addition, there is a selection method known as **elitism**, which defines the number of the best individuals at each generation, that are guaranteed to survive in the next generation. Some good individuals can be lost if they are not selected to reproduce or if they



(a) One-point crossover.

(b) Two-point crossover.

(c) Uniform crossover. With a probability of 0.5, children have approximately half of the genes from the first parent and the other half from the second.

Figure 2.14: Different kinds of crossover.

are destroyed by crossover or mutation. Many researchers have found that this technique leads to better results [Mit98].

Some of the advantages of a GA include that it [HH04]:

- Optimizes with continuous or discrete variables;
- Doesn't require derivative information;
- Simultaneously searches from a wide sampling of the cost surface;
- Deals with a large number of variables;
- Is well suited for parallel computers;
- Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum);
- Provides a list of optimum variables, not just a single solution;
- May encode the variables so that the optimization is done with the encoded variables;
- Works with numerically generated data, experimental data, or analytical functions.

The algorithm 4 shows the pseudo code of a generic GA. The mutation function receives the parameter p_m which is the probability of occurring a mutation in a chromosome.

Algorithm 4 Genetic Algorithm.

1: $population \leftarrow CreateInitialPopulation()$
2: $Evaluate(population)$
3: **while** $TerminationConditionsNotMet()$ **do**
4: [Elitism] $elite \leftarrow Elitism(population)$
5: [Selection] $parents \leftarrow Selection(population)$
6: [Crossover] $children \leftarrow Crossover(Parents)$
7: [Mutation] $mutants \leftarrow Mutation(children, p_m)$
8: **end while**

2.5 Planning Problems

Planning is the process of generating (possibly partial) representations of future behaviour prior to the use of such plans to constrain or control that behaviour. The outcome is usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents. A planning problem has an *initial state* of the world, a desired *goal state* (or set of goal states) of the world, and a *planning domain*. A planning domain is a sequence of actions which can be executed in some particular world-states, named *preconditions* and changes the world state, having a set of *effects*. A planner solves a planning problem by producing a legal sequence of actions which takes from the initial state to the final state [WKoT99, CIR99].

2.5.1 Defining a Planning problem

In order to provide a standard encoding language to describe planning problem Drew McDermott [MGH+98] released the Planning Domain Description Language (PDDL), which encourages sharing of problems and algorithms, and provides meaningful comparison of the performance of planning methods on different problems. It separates the domain description and the problem instance description in different files, so that the same domain can be used for different planning problems. The syntax is inspired by Lisp[5], so much of the structure of a domain description is a Lisp-like list of parenthesised expressions [FL03, GHK+98].

As we can see, the description of a planning problem defines a search problem: we can search from the initial state through the space of states, looking for a goal [RN10]. So, in order to solve a planning problem, AI search methods can be used. Further details are presented in the next section.

[5]List processing language invented at MIT in the 50s and developed in the 60s.

2.5.2 Problem Solving with Search Methods

A solution for a formulated problem can be seen as sequence of actions, in which the solution is one particular sequence which leads to a goal state. The sequences of actions can be stated with a *search tree*, where the root is the initial state, the branches are the actions, and the nodes represent states in the space of the problem.

The essence of search methods is the generation of the search tree. It starts in the initial state, which is the root, tests whether it is a goal state, and, if it is not, expands the current state applying each legal action, thereby generating a new set of nodes (states). After that, the method must choose which one of non-visited nodes must go next, and repeat the process until reach the goal state.

Search algorithms can be broadly divided in **uninformed** (also called blinded search) or **informed** search algorithms. The difference is that in informed search algorithms some guidance on where to look for solutions is given.

The most popular uninformed methods are *breadth-first search*, *uniform-cost search*, *depth first search*, *depth limited search* and *Iterative deepening depth-first search*.

In the scope of informed methods, *greedy best first search*, *A* search* are also very popular.

The performance of search algorithms can be evaluated with the following properties:

- **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
- **Optimality:** Does it find the optimal solution?
- **Time complexity:** How long does it take to find a solution?
- **Space complexity**: The amount of memory needed to perform the search.

Depending on the problem to solve, some of these properties can be more important than the others.

2.5.2.1 Greedy best-first search

Greedy best-first search expands first the node estimated to be closer to the goal. Thus, it evaluates nodes using just an heuristic function, $h(n)$. The heuristic gives the estimated cost of the cheapest path from the node n to the goal, so the performance of the method is highly committed to the quality of the heuristic. The algorithm is called "greedy" because at each step it tries to get as close to the goal as it can. Because of that, this method can return non optimal solutions. The greedy best-first search is also incomplete in infinite spaces. With finite states it can be incomplete in the presence of loops. However, using some technique to avoid loops, the algorithm can be complete.

2.5.2.2 A* search

A* search (pronnounced A-star search) [HNR68] is similar to the greedy algorithm method. Nevertheless, now, the evaluation function, $f(n)$, combines the heuristic, $h(n)$, with a cost-function, $g(n)$, which gives the path cost from the start node to the node n:

$$f(n) = g(n) + h(n) \tag{2.6}$$

A* is complete and, provided that the heuristic function $h(n)$ satisfies certain conditions, is **optimal** [DP85]:

- $h(n)$ is **admissible**, which means that the heuristic *never over estimates* the cost to the goal;
- $h(n)$ is **consistent**, which means that for every node n and every successor node n' generated by n with the action a, $h(n)$ is not greater than $h(n') + cost(n,a,n')$.

The pseudo-code of the search algorithm is defined in algorithm 5.

Algorithm 5 A* search algorithm.

```
1:  openList ← startNode
2:  while !Empty(openList) do
3:      n ← node in openList with the smallest evaluation
4:      if isGoal(n) then
5:          Add(closeList, n)
6:          return
7:      else
8:          succ(n) ← getSuccessors(n)
9:          for all child : Node in succ(n) do
10:             f ← Evaluate(child)
11:             if child not in closeList then
12:                 Add(openList, child)
13:             else
14:                 f' ← evaluation of child when it was previously added to closeList
15:                 if f < f' then
16:                     Add(openList, child)
17:                 end if
18:             end if
19:         end for
20:     end if
21: end while
```

2.6 Related Research

Many research related to humanoid soccer players is being developed all around the world. Namely, the humanoid biped walking is a very popular subject. Shaffi has worked

on a Truncated Fourier Series Formulation method (TFS) for gait generation, providing arm angular trajectories for a smoother and robust walking [SKAJ09]. Also he used optimization to find the best angular trajectories and optimize the TFS [SRL11].

Urieli et al. [UMK+11] has worked on the optimization of skills in a sequence. This means that, after optimizing independent skills it optimizes a skill to be performed after a specific one. For instance, if a fast forward skill is executed after a backward skill, the humanoid will lack stability on the transition. However, if after executing a backward walk it executes a slow forward walk, only after the slow forward walk it executes the fast forward walk, there won't be any stability issues.

The idea behind Urieli's work was to create different versions of the same skill: one version is a stable skill, that can be executed after any skill, the other version is an optimization of the stable version that can be used after the stable skill. Due to this optimized version of the skill, we can achieve better results with no expense on compatibility.

2.7 Conclusion

As presented in this chapter there is a lot of researching being developed in fields related to humanoid robotics. Several technologies were developed in order to help this research, including the referred simulators and humanoid robots. Nevertheless, there is still a lot of work to do. The creation of smooth and well optimized behaviours is still a very difficult task, and the RoboCup initiative has been challenging its participants to focus on this kind of problems.

Topics related to AI, such as optimization, planning problems and search algorithms were also described. These are methods highly related to the problem of this dissertation.

In robotic soccer, the humanoid players are expected to have different sequences of behaviours for the same task. However, depending on the environment, some are better than others. So, rather than having the job done, it's expected to do the job with the best approach. The hand-actions system, briefly described in 2.3.2, is not flexible enough when the goal is to reason over different ways of doing the same task. It only tries to learn them through observation and doesn't try to mix the known behaviours [OTI+00].

Chapter 3

Optimization of Behaviours

This chapter aims to describe the optimization process used in the FCPortugal Team. The optimization architecture is presented, as well as the improvements made on it. Also some related work is presented: the behaviour models were adapted in order to dynamically receive parameters and change the behaviour with regard to the environment. Section 3.4 reports the experiments made with different kinds of kicks and the obtained results.

3.1 Problem Statement

It was explained in section 2.3 how the behaviours are generated in the FCPortugal team, including the slot, step and CPG based models. Usually, the behaviours developed to humanoids aim to have the humanoid doing tasks like humans do. As referred in section 2.6 bipedal locomotion is an active area of research. Nevertheless, a football humanoid player needs to have a variety of skills: rotate, forward kick, side kick, get up, and so on. However, developing such skills for humanoid robots is a very complex task because it requires to scope many degrees of freedom and the developer has to do it joint by joint, taking care to ensure stability and efficiency.

With this, it is possible to get a skill in which the humanoid does what is expected without falling (if it was not its purpose). Yet, probably we are not getting the most from the behaviour. Maybe with some slight changes in the joint trajectories it could become more efficient. This is a very important subject, specially in a football competition like RoboCup.

3.2 Solution

The approach proposed for the described problem was to create an optimizer system in which, using some of the optimization techniques presented in section 2.4, the behaviours would experiment some changes in the joint trajectories, in order to converge into a more efficient and robuster behaviour.

These changed behaviours are then tested using the SimSpark simulator and, using some parameters like the distance achieved by the agent, whether it has fallen, among others, a fitness function gives a value with regard to the quality of the changed behaviour.

3.3 Implementation

3.3.1 Optimization System Overview

The optimizer is an extension of the work by Rei [Rei10, RRL11] and it was designed with a distributed architecture, cf. figure 3.1. It can run in several machines with several agents in order to speed up the optimization process.

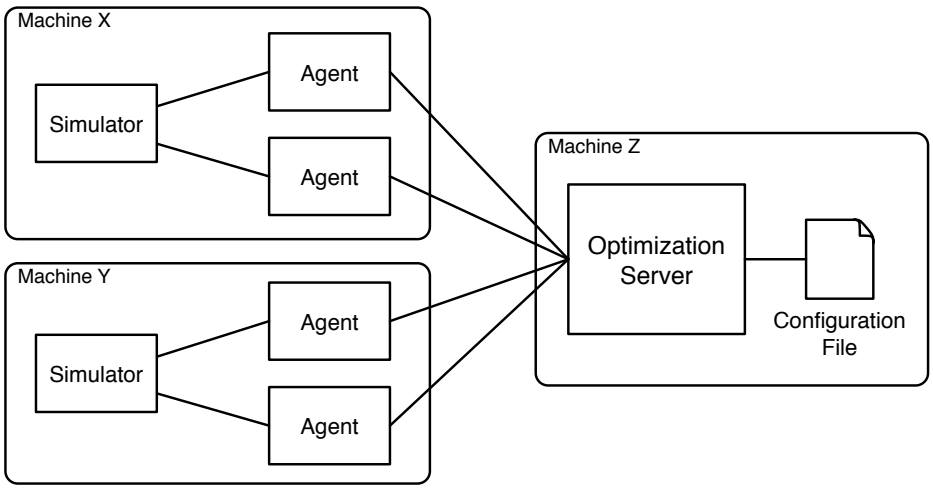


Figure 3.1: Possible configuration of the optimizer. It can run on several machines with several agents.

It is composed by the following components: optimization server, agents and simulators.

The optimization server is the most important and it's responsible for controlling the other components. This is, it's the optimization server which starts and stops the simulators and the servers and tells the agents what to experiment. The agents are the guinea

pigs[1]: they only receive the modified behaviours, execute them, collect the relevant information about the execution and reply it to the agent. The simulator is simply the habitat: it's required in order for the agents to execute.

The execution of the optimization system works like this:

1. The optimizer server is started by the user.
2. The optimization server reads the given configuration file and executes the specified scripts for starting both the simulator and the agents. The started agents receive in their arguments list some necessary information including the location (IP address) of the server.
3. The server starts the optimization algorithm (e.g. Hill Climbing, Tabu Search, etc.), and waits for connections from the agents.
4. The agent connects the server and receives the behaviours it has to execute.
5. After the execution, the agent collects the information about the experiment (ball distance, time elapsed, and so on) and sends it back to the server being ready for a new experiment.
6. The server receives the results from the agent and executes the fitness function.
7. The server processes the result accordingly with the optimization algorithm and waits for other agents.

In order to have several agents executing concurrent experiments, the server creates a thread for each agent. This thread creates a TCP server and, after the agent's connection, provides it with the behaviour, receives the experiment results data, and calculates the fitness.

3.3.2 Experiments Execution

The optimization server sends to the agent the information necessary to execute the experiment, including the modified behaviour and some parameters about how to execute the experiment.

Figure 3.2 describes the agent/optimization server interaction. Basically, the agent executes the experiment as follows:

1. Connect to the server and receive the experiment's data;
2. Prepare the experiment. According with the received data the agent has to set its position, set the ball's position.

[1] A guinea pig is a pet typically used for laboratory research.

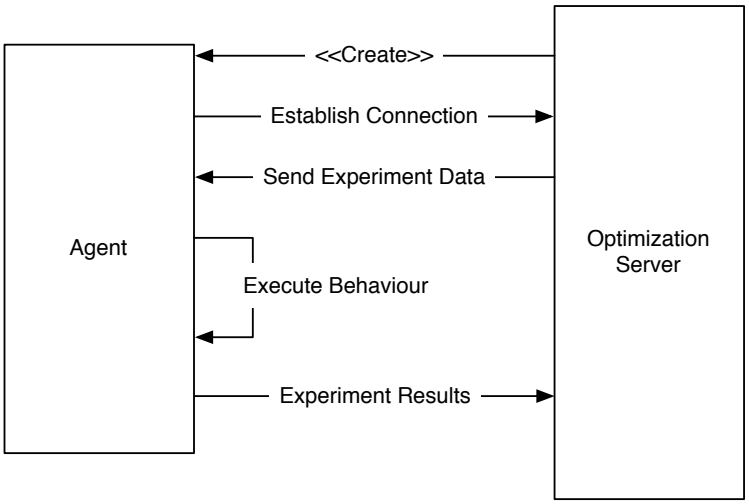


Figure 3.2: The optimization process - communication between the agent and the server.

3. Wait. After preparing the experiment probably the agent is not ready to start it. For instance, immediately after beaming, the agent is not very stable, so it's necessary to wait a short period of time until it is ready.

4. Execute the received behaviour.

5. Wait again. This is necessary in order to collect the data. For instance, if we are optimizing a kick, we want to wait until the ball stops in order to have a reliable information about the achieved distance.

6. Collect the results and send it to the server.

3.3.2.1 Gyroscope Stabilization

The time the agent has to wait before and after the experiment is determined in the configuration file. Alternatively, instead of determining the time the agent has to wait before the experiment, there is an option in which the agent waits until is stable, reading the information from the gyroscope.

When this option is set on the agent will wait until receive from the gyroscope data values in the range of $]-5.0, 5.0[$. Then, the agent is ready to execute the behaviour.

This is illustrated in the plot of the figure 3.3. The lines in the colours blue, green and red represent the values got from gyroscope in the X, Y and Z axis, respectively. The gray area is the range in which the optimizer considers the agent as being stable and ready to perform the experiment, *i.e.* values between -5 and 5 in all axis at the same time.

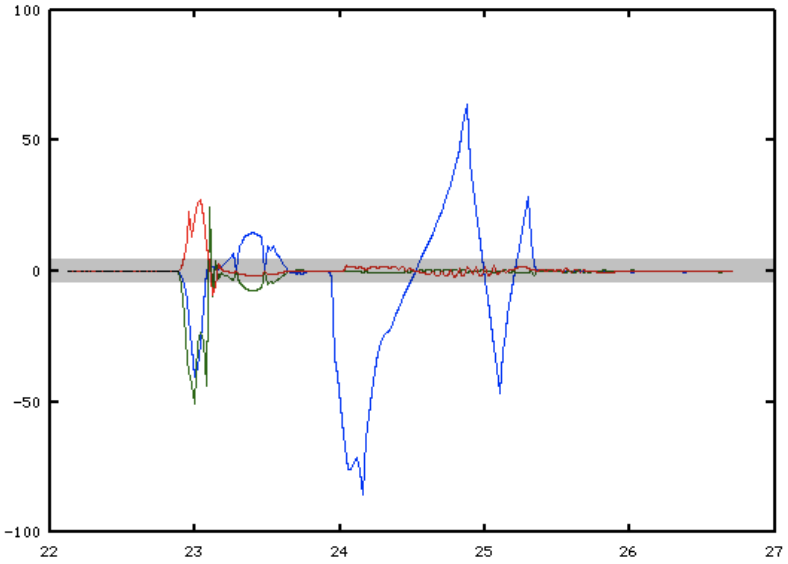


Figure 3.3: Values read from the agent's gyroscope at the execution of simulator's *Beam* action.

3.3.3 Experiments Evaluation

In order to evaluate the experiment, the agent collects some data and sends it to the server.

The collected data includes:

- The time taken to execute the behaviour;
- A boolean indicating whether the agent fell;
- The agent's initial position;
- The agent's final position;
- The distance travelled by the agent in each axis;
- The final agent's orientation;
- The ball's initial position;
- The ball's final position;
- The distance travelled by the ball in each axis;
- The simulation time.

After receiving this data, the server needs to evaluate the experiment. This is, for each experiment the server needs to give a value allowing to compare different experiments and identify which one is better. This is made calling the objective function (or fitness function).

There are two ways to specify an objective function (or fitness function). The first is by implementing a C function and include it in the source code of the optimizer. The drawback of this approach is that whenever it is needed to create an objective function, some changes have to be made in the code. This requires some knowledge about the implementation code. Besides, it is necessary to add a string identifier to the new function, so that it is possible to choose it with a string in the configuration file.

So, to simplify the optimization configuration, it's desirable to have a manner of specifying the objective function outside the optimizer core, in an external file. This can be made using script files, in which the objective functions is implemented. This requires to parse a function

To handle this drawbacks, it was used a scripting language known as Lua [IDFF96, IdFC07]. Lua is an embeddable scripting language with a simple C API and it has been widely used as a tool to customize applications. It has the usual control structures (whiles, ifs, etc.), function definitions with parameters and local variables, it also provides functions as first order values, closures, and dynamically created associative arrays (called tables) as a single, unifying data-structuring mechanism [Cam02].

Using the Lua API, the objective function for the optimization can be implemented as a script in a separated file and it is only necessary to specify the path to it.

3.3.4 The Configuration File

In order to make this optimizer easier to use in different situations, a configuration file is used to specify the characteristics of the optimization and some parameters about how the agent will execute the experiment. This also avoids having to recompile the optimizer any time we want to change a parameter.

To process the files a simple library is used, *libconfig*[2] [Mou11], which is a library specially designed for reading, manipulating and writing configuration files. For this kind of configuration it's more compact and readable than XML, and it is type-aware, which means that it is not necessary to do string parsing, unlike XML.

The most relevant parameters are the following:

- **behavior** - Path to the XML specification file of the behaviour to be optimized.
- **type** - Type of behaviour. It can be "StepBehavior", "SlotBehavior" or "CPGBehavior".
- **objectiveScript** - File path to the LUA script with the implemented method "objective", which evaluates the executed experiment and returns its fitness.

[2]Further information in: http://www.hyperrealm.com/libconfig/

- **objective** - Instead of using a LUA script, a method can be implemented in the code. This is not recommended because it's necessary to be familiarised with the optimisation source code and most of the fitness functions can be easily specified with a LUA script. However for a fitness function which handles more complex data this might be necessary.

- **script** - File path to the script responsible for starting the agent and the simulator. **name** - Name for the resulting optimised behaviour.

- **algorithm** - Algorithm that is going to be used in the optimisation. The available options are "hc" (Hill-Climbing), "hcr" (Random Hill-Climbing), "sa" (Simulated Annealing), "ga" (Genetic Algorithm) and "tabu" (Tabu Search).

- **numExp** - Number of executions for the same experiment.

- **nMax** - the number of iterations for the algorithms main loop.

- **numThread** - If more than one agent can execute the experiments, the optimisation can be made in less time. In this option are specified the number of agents that will be executing.

- **waitBefore** - Time in seconds that the agent will wait before executing the experiment. This needs to be specified because after the agent being beamed it leaks some balance. If the behaviour is executed without waiting, probably it won't be well executed and maybe the agent will fall.

- **useGyroStabilization** - Instead of using the waitBefore option, another approach is possible. Using the data received from the agent's gyroscope, it's possible to know if the agent has enough stability to start the execution. Further details about how this works in section 3.3.2.1. The value in this parameter is a boolean (true/false).

- **waitAfter** - The time in seconds, in which the agent will wait before collecting the data about the experiment to be evaluated by the fitness function. This is useful, for instance when we are trying to optimise a kick and we need to know the final distance of the ball. If we check the ball distance immediately after the execution of the kick behaviour, probably the ball would still be moving.

- **beamBall** - The position in which we want the ball in the start of each execution. Again, if we want to optimise a kick after the execution it's expected that the ball won't be in the same position, so we need to reset the position of the ball.

- **algorithm parameters** - the parameters of the algorithm such as the population size for genetic algorithms or the size of the tabu list for a *Tabu Search*.

- **behaviour optimization parameters** - the specific parts of the behaviour to be optimized such as which slots or steps for Slot Behaviours or Step Behaviours, respectively, which joints and restrictions to enforce such as symmetries between joints.

- **minChange** - minimum value to change an optimization variable by adding to it to obtain a neighbouring solution (this should be a negative number) - does not apply to time intervals (e.g. slot execution time).

- **maxChange** - maximum value to change an optimization variable by adding to it to obtain a neighbouring solution (this should be a positive number) - does not apply to time intervals (e.g. slot execution time);

- **minChangeDelta** - minimum value to change an optimization variable that represents a time interval by adding to it to obtain a neighbouring solution (this should be a negative number);

- **maxChangeDelta** - maximum value to change an optimization variable that represents a time interval by adding to it to obtain a neighbouring solution (this should be a positive number).

- **serverRestartTime** - maximum amount of time the simulator can run without being restarted (in seconds). It has to be less than the full time of a simulation game, because after that the game stops and the experiment can be compromised. Further details in section 3.3.6.

3.3.5 Implemented Methods

The implemented methods were from Rei's work [Rei10, RRL11] and they are namely: **Hill Climbing** (HC), **Tabu Search** (TS), **Simulated Annealing** (SA) and **Genetic Algorithm** (GA). The methods were already generically explained in the section 2.4. This section aims to present some details about their implementation.

3.3.5.1 Hill Climbing

This is the simplest method available in the optimizer. There are two variations of this method. One variation is known as *Hill Climbing* (HC), changes an optimization variable by a value that varies with a constant factor – the *acceleration* – increasing or decreasing whether the solution is better or not. The other variation *Hill Climbing Random* (HCR) in which the optimization variable is changed by a random value.

3.3.5.2 Tabu Search

The implemented TS method is very similar to the Hill Climbing Random variation. The difference is that it mantains a list of the tested solutions. So whenever a new solution is generated, its existence in the list is verified. If it already exists, the solution is thrown away. To use this method the configuration file needs to have an extra statement, the *listsize*, which tells the maximum size of the list. If, whenever a new solution is calculated, the list is full, this new solution will replace the oldest in the list.

3.3.5.3 Simulated Annealing

The SA method is a local search algorithm which implements a mechanism to escape from a local minimum that is not the global minimum.

The extra parameters to specify in the configuration file are:

- *temperature* - initial temperature;
- *cool* - temperature decreasing rate at every new solution;
- *saRestart* - maximum number of iterations without finding a new bestSolution. After those the search is restarted at the current best solution.

3.3.5.4 Genetic Algorithm

The GA was implemented with the *elitism*, *crossover*, and *mutation* features. The *selection* made in every iteration uses the **roulette-wheel selection**. This works by giving a ranking the individuals by its fitness and select individuals with a probability with regard to their ranking, as illustrated in figure 3.4. The adopted method for crossover is the *uniform crossover*.

The extra parameters to specify in the configuration file in order to use GA are:

- *nElite* - number of elite individuals in each iteration;
- *popSize* - size of the population;
- *pMutate* - probability of mutation for a gene;
- *pCrossover* - probability of a locus to be selected for crossover.

3.3.6 RoboCup Server Modifications

Whenever an experiment is made, the agent's and ball's positions changes. If we want to have the same conditions for every execution, it's necessary to undo these changes. In order to have a fully automated optimization, it is necessary to have an automated way of

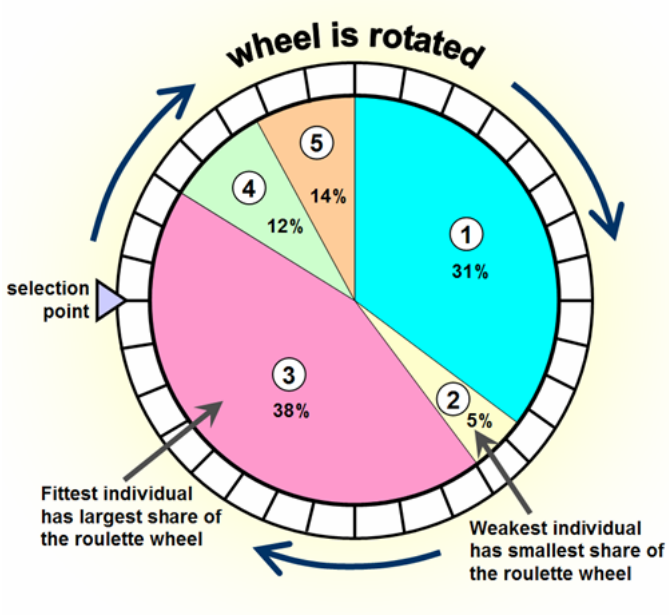


Figure 3.4: Roulette wheel approach: based on fitness (source: [new])

controlling the agent's and ball's positions. So, a few changes were made in the simulation server.

The simulator has a feature which allows the creation of plugins. These plugins are actions which change the simulation environment and they are triggered when the server receives specific messages. The simulator already had a plugin that allowed the repositioning of the agent. This is called the *Beam* action. However, this can only be done in the beginning of the games. The optimizer needs to do this in every game state, so a few changes were made in the *Beam* plugin.

To the repositioning of the ball, a similar plugin was created.

Another modification was to allow the game to automatically start. Originally the server would only manually start it. It was also modified the duration of a game. In the original simulator, the server had to be reinitialized after less than 600 seconds of simulation, which took approximately 20 seconds. This is acceptable, but in order to speed up the optimization, this value was changed to a larger number.

3.4 Experimental Results

3.4.1 Optimization of a Side Kick

A good approach for developing behaviours, is to start by creating a slow and not so robust version of the behaviour by hand. After that, the optimiser can be used to create a new version which is faster and more robust.

In this experience, it was developed a side kick using the Slot Behaviour model. The execution of the non optimized behaviour can be seen in figure 3.5.

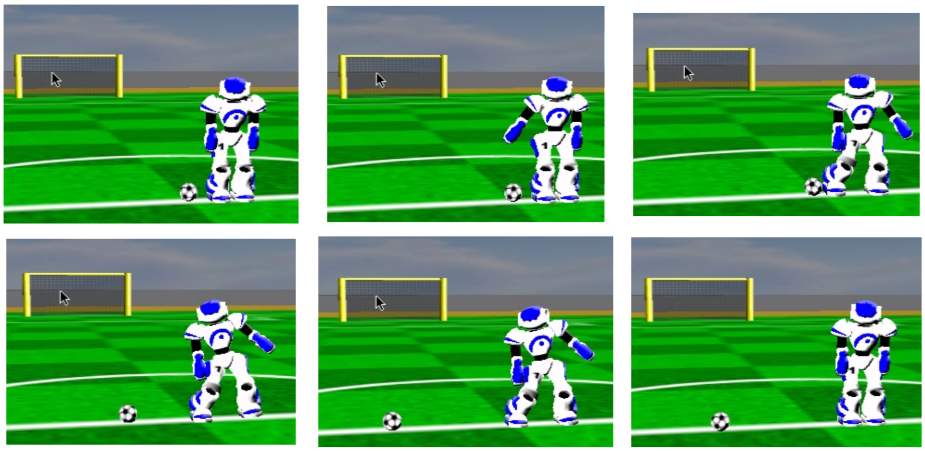


Figure 3.5: Sequence of images showing the execution of the side kick skill (not optimized).

3.4.1.1 Optimization Setup

The side kick behaviour is composed by 6 slots, in which the first and the last serve to put the agent in a standard pose, i.e. to reset the agent pose. These reset slots don't need to be optimizez because they won't interferer with the behaviour itself. With this, there are 17 optimization variables left, consisting of joint angles and slot deltas.

According with Rei's work [RRL11], the Hill Climbing Random and Tabu Search algorithms are the methods which led to the best results. So, in this experiments these methods were used.

In a side kick the main objective is to get higher ball distances in less time and also not to have the ball going front or back. So, the fitness function awards solutions with these characteristics.

$$f(x) = \frac{bdist_y^3 * bell(bdist_x)}{\Delta t} \tag{3.1}$$

The function $bell(x)$ is the probability density function of a normal (or Gaussian) distribution and is known as the *Gaussian function* or the *bell function*:

$$bell(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.2}$$

It is used in the fitness function with $\mu = 0.0$ (a.k.a. *mean*) and $\sigma^2 = 1.0$ (a.k.a. *variance*) with the aim of giving better fitness values to solutions in which the ball goes strictly to the side. The figure 3.6 shows the graph of the bell function.

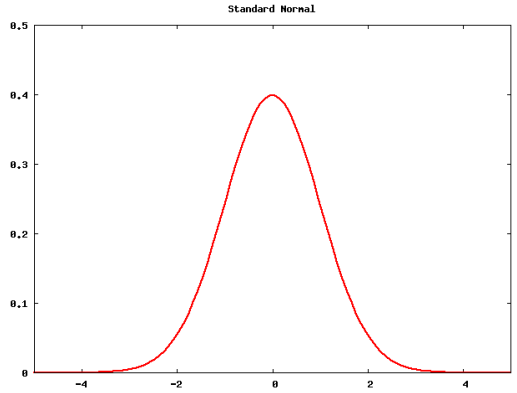


Figure 3.6: The classic bell curve with $\mu = 0.0$ and $\sigma^2 = 1.0$, i.e. the *standard normal*.

The optimization parameters used for the experiment are shown in the table 3.1. The number of threads (or agents) is 1 because each simulation only have 1 ball.

3.4.1.2 Experimental Results

The results achieved in the optimization can be seen in the table 3.2. The best result reaches 3.07 meters of ball distance, executes in 0.88 seconds and was obtained with the Tabu Search algorithm. The total time is approximately the same in both methods because the elapsed time depends mostly on the duration of each experiment.

Table 3.3 presents the achieved solutions with all the optimization variables. Slots 0 and 5 were omitted because they were not optimized.

The graphs of figure 3.7 present the evolution of the fitness score of the best and current solutions over the iteration, for both Hill Climbing and Tabu Search methods. We can notice that, over the iterations, the fitness of the solution that is being evaluated varies greatly, despite the very slight differences. Even though the Tabu Search only reached the best fitness at iteration 404, a solution with the same fitness as the best solution in Hill Climbing, was achieved in less time at iteration 63.

3.4.2 Optimization of the Forward Kick for Maximum Distance

In this experiment the optimization was used to improve the FCPortugal team's longest forward kick in order to reach even higher ball distances. Again, both Hill Climbing and Tabu Search methods were used.

3.4.2.1 Optimization Setup

The original behaviour is a slot based behaviour with 10 slots, c.f. figure 3.8. Again, the first and last slots serves to reset the humanoid's pose. So, this time we have 8 slots to optimize, leading to 61 variables of optimization.

Table 3.1: Side kick optimization parameters.

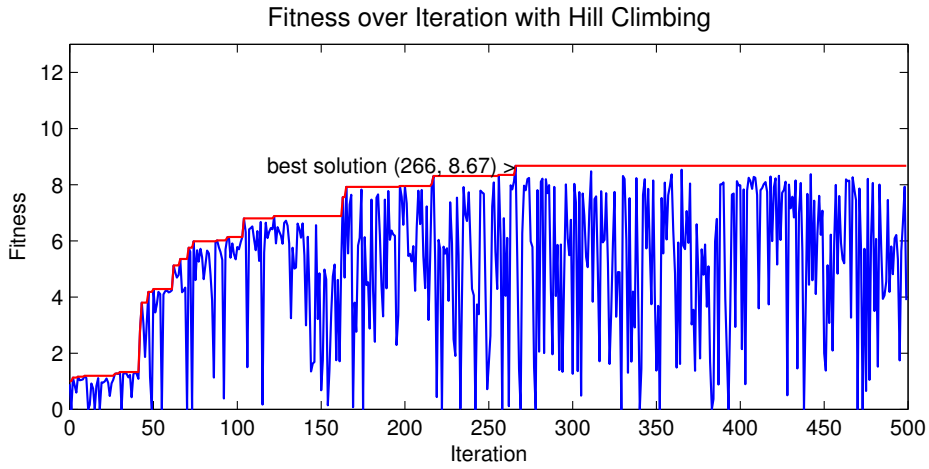
Parameter	Value
Number of experiments	5
Number of threads	1
Minimum Change for Angles	-5.0
Maximum Change for Angles	5.0
Minimum Change for Deltas	-0.3
Maximum Change for Deltas	0.3
Number of Iterations	500
Tabu List Size (Tabu Search)	1000
Wait for Gyroscope Stabilization	True
Time to Wait After the Experiment (sec)	3.0
Beam Ball Position	(0.0, 0.0)

Table 3.2: Side kick optimization results.

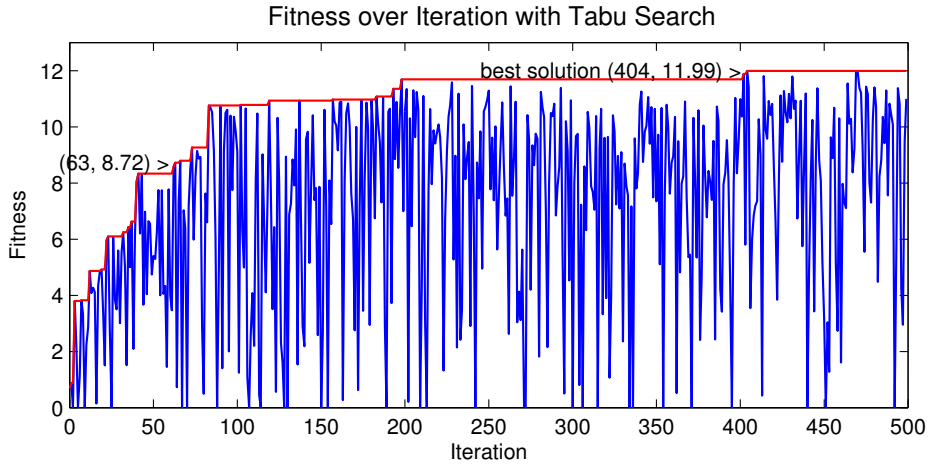
	Ball Distance (m)	Duration (sec)	Best Solution (iteration)	Best Solution Time (simulation sec)	Total Time (simulation sec)
Original	1.42	1.14	-	-	-
Hill Climbing	2.75	0.89	266	9756	18042
Tabu Search	3.07	0.88	404	14553	17920

Table 3.3: Solution values of the optimization variables.

	Original	HCR	TS
Slot 1			
Delta	0.07	0.03246	0.07
Slot 2			
Delta	0.4	0.15446	0.14428
Joint 5 (rleg2)	-40.0	-45	-47.1363
Joint 7 (rleg3)	35.0	35	35.3528
Joint 9 (rleg4)	-70.0	-71.0269	-70.1226
Joint 11 (rleg5)	35.0	36.9569	35
Joint 12 (lleg6)	32.0	25.0	24.7303
Joint 13 (rleg6)	24.0	22.0454	21.0467
Joint 15 (rarm1)	-90.0	-90.0	-97.763
Joint 16 (larm2)	40.0	38.3645	51.5218
Joint 17 (rarm2)	-10.0	-6.40997	4.65801
Slot 3			
Delta	0.05	0.05	0.05
Joint 6 (lleg3)	30.0	36.8193	35.0363
Joint 8 (lleg4)	-70.0	67.78888	-67.2204
Joint 10 (lleg5)	40.0	41.6722	40.3109
Joint 12 (lleg6)	-5.0	-1.35246	1.4933
Slot 4			
Delta	0.4	0.04513	0.0



(a) Hill Climbing.



(b) Tabu Search.

Figure 3.7: Graphs of the fitness over the iteration reported in the optimization of the side kick skill for both Hill Climbing and Tabu Search algorithms. The blue lines represent the fitness of the current experiment and the red lines are the best fitness until that iteration.

In this forward kick the main objective is to get higher ball distances in less time and with the ball going straight to the front, this is, without having the ball going too much to the left or to the right). So, the fitness function is as follows:

$$f(x) = \frac{bdist_x^3 * bell(bdist_y)}{\Delta t} \tag{3.3}$$

The fitness function uses the bell function stated in equation 3.2 to award solutions in which the ball describes a strictly forward trajectory.

The table 3.4 shows the optimization parameters used in the experience. Comparing with the parameters of the side kick optimization (c.f. section 3.4.1) the time to wait after each behaviour's execution has increased from 3.0 to 4.0 seconds. This is necessary

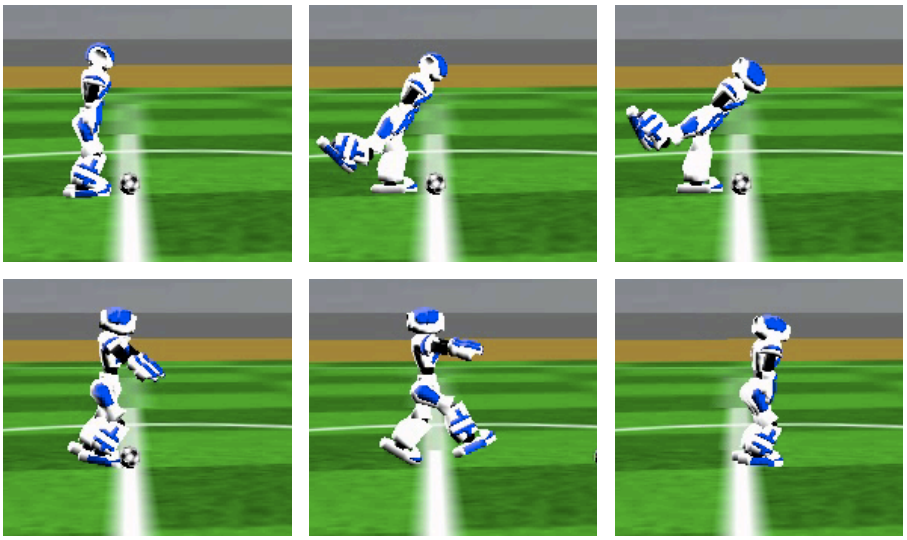


Figure 3.8: Sequence of images showing the execution of the front kick skill (not optimized).

because now the ball is reaching higher distances, so it will take longer to stop.

3.4.2.2 Experimental Results

The achieved behaviours are presented in table 3.5, where it's possible to compare the optimization variables. The results of this experiment are reported in the table 3.6.

Now, the best results were achieved with the Hill Climbing method resulting in a kick that can move the ball 6.30 meters forward. Also, the duration of the skill was reduced from 2.04 seconds to 1.78 seconds.

The graphs of figure 3.9 shows the convergence of the optimization methods.

3.4.3 Optimization of the Forward Kick for Different Distances

In order to have a high-level decision for the FCPortugal team's humanoid, this is, with tactical decision and capable of doing cooperation moves, it is very important to have a broad panoply of skills that provides flexibility to every situation. For instance, to pass the ball it is important to kick the ball for different distances. If a teammate is at a distance of 4 meters it's not a good idea to kick the ball with the same power as if it was 6 meters.

So, the idea of this experiment is to optimize the forward kick, described in section 3.4.2, to create new forward kicks for different distances: 6, 5 and 4 meters.

3.4.3.1 Optimization Setup

Starting on the 6 meters kick, both Hill Climbing and Tabu Search methods were used. For the following kicks, namely 4 and 5 meters kick, only the best method is used. As

Table 3.4: Forward kick optimization parameters.

Parameter	Value
Number of experiments	6
Number of threads	1
Minimum Change for Angles	-5.0
Maximum Change for Angles	5.0
Minimum Change for Deltas	-0.3
Maximum Change for Deltas	0.3
Number of Iterations	500
Tabu List Size (Tabu Search)	1000
Wait for Gyroscope Stabilization	True
Time to Wait After the Experiment (sec)	4.0
Beam Ball Position	(0.0, 0.0)

Tabu Search has proved to be better than Hill Climbing on these kind of behaviours, only Tabu Search was used on these experiments.

The optimizer configuration is presented in the table 3.7. A similar configuration as the previous experiments configuration was used.

However, the fitness function has suffered some modifications:

$$f(x) = \frac{bell_{(\mu=d,\sigma^2=1.0)}(bdist_x)^2 * bell_{(\mu=0.0,\sigma^2=1.0)}(bdist_y)}{\Delta t} \tag{3.4}$$

There are 2 main differences:

- Now the distance of the ball in the x-axis is raised to the power of 2, instead of 3. This happens to decrease the emphases on the ball distance over the time and the precision of the kick in y-axis (lateral deviation), which is desired to be near 0.

- The bell function (c.f equation 3.2) is also used with the ball distance in x-axis. However it is now applied with $\mu = d$, where d is the distance of the desired distance of the ball (6, 5 or 4 meters).

3.4.3.2 Experimental Results

The results can be seen in the table 3.8. Comparatively to the previous results tables this has a new column, called *Lateral Deviation*, because precision is considered an important feature for the desired kicks.

As the best optimization algorithm for the 6 meters kick was the Hill Climbing method, only this algorithm was used for the 5 and 4 meters. Nevertheless, according to the graph of figure 3.10, both methods had a similar behaviour. However, in the longest distance

Table 3.5: Solution values of the optimization variables.

	Original	HCR	TS
Slot 1			
Delta	0.24	0.11469	0.24
Slot 2			
Delta	0.16	0.16	0.16
Joint 7 (rleg3)	21	22.6999	18.3649
Joint 9 (rleg4)	-39	-39	-39
Joint 11 (rleg5)	30	29.5281	30
Slot 3			
Delta	0.36	0.3387	0.36
Joint 2 (lleg1)	0.285636	0.3387	0.28564
Joint 3 (rleg1)	0	0	0
Joint 4 (lleg2)	-12	-13.0721	-12
Joint 5 (rleg2)	-11	-11	-11
Joint 6 (lleg3)	35.7833	35.783	35.7833
Joint 7 (rleg3)	-23.1867	-23.1867	-26.22
Joint 8 (lleg4)	-0.131957	0.13196	-0.13196
Joint 9 (rleg4)	-70.4176	-70.4176	-61.6771
Joint 10 (lleg5)	10.5019	6.992767	10.5019
Joint 11 (rleg5)	74.755	74.755	72.3305
Joint 12 (lleg6)	12	12	12
Joint 13 (rleg6)	11	11	11
Joint 14 (larm1)	-90	-90	-90
Joint 15 (rarm1	-90	-90	-90
Slot 4			
Delta		0.02	0.02
Joint 2 (lleg1)	0.951197	-8.1056	0.17739
Joint 3 (rleg1)	1	1	1
Joint 4 (lleg2)	-12	-12	-12
Joint 5 (rleg2)	-11	-8.8774	-11
Joint 6 (lleg3)	-20	-20	-20
Joint 7 (rleg3)	20	20	20
Joint 8 (lleg4)	-21.013	-21.013	-21.013
Joint 9 (rleg4)	-120	-120	-120
Joint 10 (lleg5)	10.5019	10.5019	15.814
Joint 11 (rleg5)	58	60.6009	58.6318
Joint 12 (lleg6)	12	12	12
Joint 13 (rleg6)	11	11	11
Joint 14 (larm1)	0	0	0
Joint 15 (rarm1)	0	0	0

	Original	HCR	TS
Slot 5			
Delta	0.04	0.04	0.04
Slot 6			
Delta	0.02	0.02	0.02
Joint 2 (lleg1)	0	0	0
Joint 3 (rleg1)	1.5039	1	1
Joint 4 (lleg2)	-12	-13.8203	-12
Joint 5 (rleg2)	-11	-11	-11
Joint 6 (lleg3)	-20	-24.7031	-20
Joint 7 (rleg3)	89	89	89
Joint 8 (lleg4)	-20	-20	-20
Joint 9 (rleg4)	-1	0.31748	-1
Joint 10 (lleg5)	10.5019	10.5019	10.5019
Joint 11 (rleg5)	-45	-44.0853	-430032
Joint 12 (lleg6)	12.622	12.622	12.622
Joint 13 (rleg6)	11	11	7.47145
Slot 7			
Delta	0.16	0.16	0.16
Slot 8			
Delta	0.18	0.07343	0.18
Joint (lleg1) 2	0	0	0
Joint 3 (rleg1)	0	0.6688	0
Joint 4 (lleg2)	0	0	0
Joint 5 (rleg2)	0	0	0
Joint 6 (lleg3)	45	44.3704	45
Joint 7 (rleg3)	0	0.0	2.09151
Joint 8 (lleg4)	-90	-90	-90
Joint 9 (rleg4)	0	-4.84362	1.01388
Joint 10 (lleg5)	45	45.0	45
Joint 11 (rleg5)	0	-2.72108	0
Joint 12 (lleg6)	0	0	0
Joint 13 (rleg6)	0	0	0
Joint 14 (larm1)	-90	-84.1491	-85.056
Joint 15 (rarm1)	-90	87.4989	-90
Joint 16 (larm2)	0	0.0	0
Joint 17 (rarm2)	0	-3.12064	-2.39896
Joint 18 (larm3)	0	0	0
Joint 19 (rarm3)	0	0	0
Joint 20 (larm4)	0	1	0
Joint 21 (rarm4)	0	-1	0

Table 3.6: Forward kick optimization results.

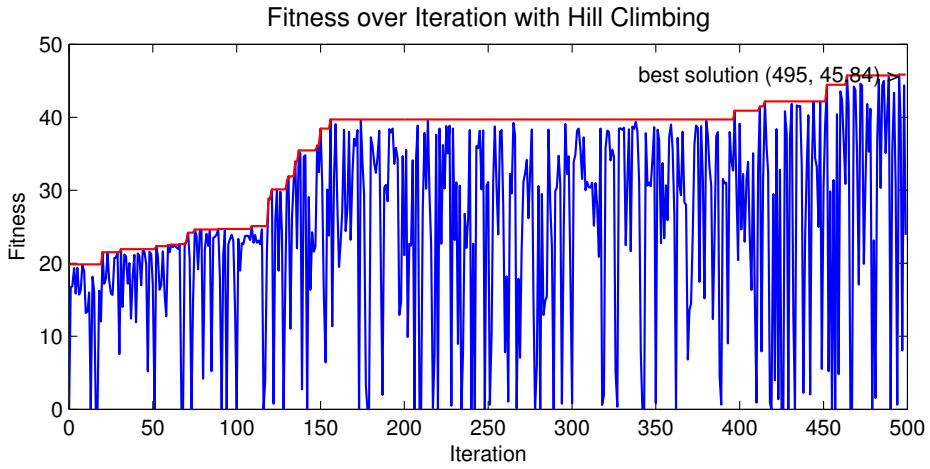
	Ball Distance (m)	Duration (sec)	Best Solution (iteration)	Best Solution Time (simulation sec)	Total Time (simulation sec)
Original	5.15	2.04	-	-	-
Hill Climbing	6.30	1.78	495	27419	27637
Tabu Search	5.65	2.04	356	20159	28258

Table 3.7: Different distances forward kicks optimization parameters.

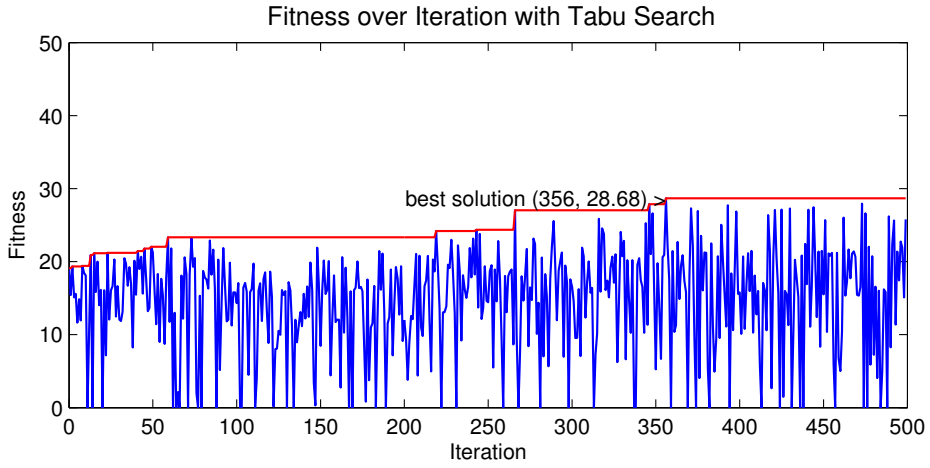
Parameter	Value
Number of experiments	5
Number of threads	1
Minimum Change for Angles	-5.0
Maximum Change for Angles	5.0
Minimum Change for Deltas	-0.3
Maximum Change for Deltas	0.3
Number of Iterations	500
Tabu List Size (Tabu Search)	1000
Wait for Gyroscope Stabilization	True
Time to Wait After the Experiment (sec)	4.0
Beam Ball Position	(0.0, 0.0)

Table 3.8: Different distance kicks optimization results.

	Ball Dist. (m)	Duration (sec)	Lateral Deviation (m)	Best Sol. (iteration)	Best Sol. Time (sim. sec)	Total Time (sim. sec)
Initial	6.30	1.78	0.65	-	-	-
6 meters (TS)	6.15	1.80	0.52	169	7683	22673
6 meters (HCR)	6.15	1.74	0.53	475	21593	22682
5 meters (HCR)	5.13	1.76	0.46	168	7604	22478
4 meters (HCR)	3.87	1.71	0.01	232	10382	22963



(a) Hill Climbing.



(b) Tabu Search.

Figure 3.9: Graphs of the fitness over the iteration for the longest distance kick optimization with the Hill Climbing and Tabu Search methods. The blue lines represent the fitness of the current experiment and the red lines are the best fitness until that iteration.

kick optimization, Hill Climbing also lead to the best results (c.f. section 3.4.2.2) which enforces this choice.

In the figure 3.11 the distances of the best solution over the optimization iteration is presented for the 6, 5 and 4 meters. Unlike the fitness function, the distance function is not monotonic. This happens for two reasons: first, the fitness function is given not only by the distance achieved but also by the duration of the execution and the lateral deviation (ball distance in the y-axis); and second, because the value of distance is not being optimized but its deviation to a certain value.

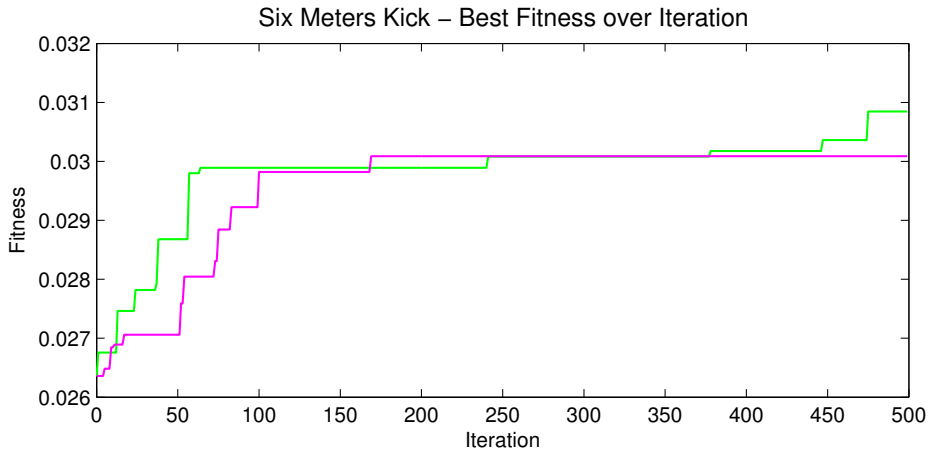


Figure 3.10: Graph comparing the convergence of both Hill Climbing (green line) and Tabu Search (pink line) algorithms.

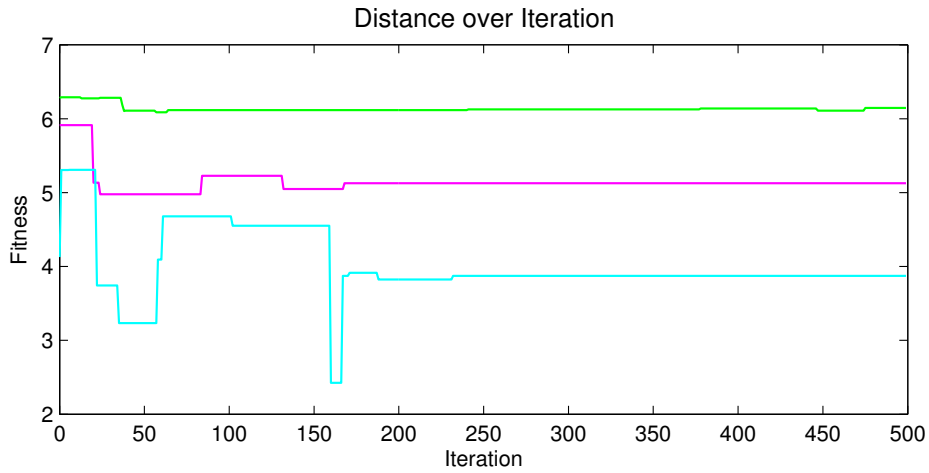


Figure 3.11: Graph of the distance over the iteration for the Hill Climbing optimization. The green line is the 6 meters function, the pink is the 5 meters and cyan is the 4 meters.

3.5 Summary

In this section were described the optimization system developed and some results achieved with it. The Side Kick was optimized, reaching the best results with the TS algorithm, making possible to reach higher values in the ball's distance in a shorter time period.

The Forward Kick was optimized to reach the maximum possible ball distance, and again good results were achieved: the kick that used to reach a ball distance of 5.15 meters, currently does 6.30 meters (an improvement of 22%). Surprisingly, this results were achievied with Hill Climbing algorithm, instead of the expected Tabu Search. Probably this happens due to the method of checking wether a solution has been already tested, i.e. whether a solution is in the tabu list. The adopted method to compare behaviours is based on a sum of the differences between the current behaviour and a behaviour on the tabu list. If this sum is under a certain value, both behaviours are considered to be equal.

As it was noticeable, a slightly difference in the forward kick behaviour, can make a big difference in its results. It is possible that in this case, different solutions, that could lead to better results, are not distinguished and are considered as being inside of the tabu list and unfortunately they are discarded.

New kicks were also generated using optimization, providing to the humanoid agent, the capability of kicking the ball with different distances, namely 4, 5 and 6 meters.

Chapter 4

Humanoid Behaviours Planning

4.1 Problem Statement

The FCPortugal agent has a wide range of behaviours. In order to play soccer it has to combine different behaviours. However, most of the times, the same task can be done with many different sequences of behaviours.

For instance, when the agent needs to reach a certain position it can execute a forward walk, a side walk or even a back walk, rotating whenever it is needed. However, not all the options provide the faster execution for the agent.

So it's very useful to have an approach that could combine all the available behaviours, and, with regard to the environment, choose the best sequence of behaviours to execute, being able to avoid obstacles, and reach a certain position with a certain orientation.

4.2 Solution

To overcome this problem, the idea is to solve it as a planning problem (c.f. section 2.5). Then, using an informed search algorithm, the sequences of behaviours can be stated as a search tree, in which the nodes are the states and each node can be expanded with the available actions, according to some preconditions.

This section presents the model used to describe the actions and the state of the world.

4.2.1 States

The states can be described with the agent's position, the agent's orientation and the ball's position. So, to start the planning it's only required to give **initial state**, with the current information perceived by the agent, and the **goal state**, which is the desired state according to the FCPortugal's strategy. A possible and simple planning problem is to set the

final agent's position as the current ball position and the final orientation as the angle of the line from the ball to the opponent goal.

4.2.2 Planning Domain

The planning domain consists in the set of available behaviours. It can be a forward walk, a rotate, and so on. These actions can be described by their effect in the agent's position and orientation and in the ball's position.

With this information it is possible to predict the effect of an action in a given state. Each action will also have a measure of the duration of its execution.

4.2.3 Evaluating Function

The evaluating function $f(x)$ is a combination of the cost $g(x)$ and the heuristic function $h(x)$:

$$f(x) = g(x) + h(x) \tag{4.1}$$

The **cost function** of a node can be measured using the duration of the sequence. This can be calculated by the sum of the duration of all actions in the path from the root of the search tree (initial state) to the given node.

The **heuristic function**, which gives a predicted value for the distance of a state to the solution, has to deal with different stages of the desired task. We have to consider that if the ball is not in the target position, the agent needs to first reach the ball and, only after moving the ball to the right position, reach its objective position.

So, if the ball is not on the objective position, the heuristic must scope the agent's distance to the ball, then the ball's distance to the ball's goal position and finally the distance between the initial ball position and the goal position of the agent. Figure 4.1 illustrates this situation. Point **1** represents the agent's initial position, **2** is the agent's intermediate position and **3** is the agent's final position. Point **I** is the Ball's initial position and **II** is the ball's final position. The final state is represented with the points **3** and **II**, so the heuristic has to scope the distances *d1*, *d2* and *d3*.

Another information to consider in the heuristic is the orientation. If the agent is far from the target, it is not a very important information, because first it's important to reach the target. However, if the agent is near to the target, it's important to consider the difference of the orientation to the goal orientation. This is the same when the agent needs to reach the ball to kick it.

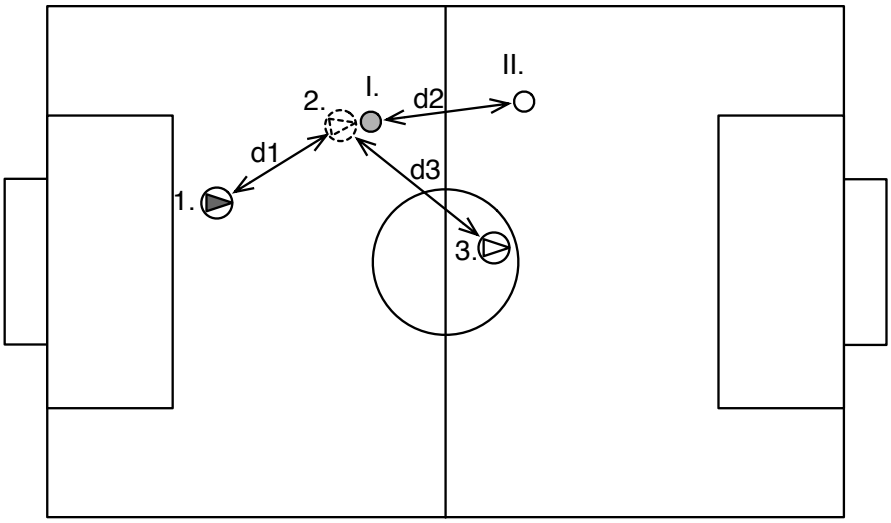


Figure 4.1: Distance from the initial state to the goal.

4.3 Implementation

This section aims to show a description about the implementation details of the solution.

4.3.1 Design Overview

The solution was designed as a component of the FCPAgent, as illustrated in figure 4.2. So, the planner is called by the agent in its *think* routine, which is the main routine executed in each simulation cycle.

The three main components are presented in the diagram of the figure 4.2: the *Planning Engine*, the *Search Algorithm*, and the *Skills Manager*.

The *Planning Engine* is responsible for:

- Read and process the planning domain description file;
- The evaluation function;
- The expansion of the states with the available actions, according with the preconditions.

The *Search Algorithm* is a reused component from Justin Heyes-Jones [1], and it provides a standard implementation in *C++* for the A* search algorithm, which can be used for the greedy algorithm also.

The Skills Manager is a component created for this purpose which didn't exist in the agent, but it's also very useful in other problems. It is a component which is responsible

[1] Source available at: http://www.heyes-jones.com/astar.html

for managing all the agent's behaviours. It stores the behaviours in a map container, allowing to dynamically return a behaviour by its name in $O(\log(n))$. This is needed by the planner, because the result is given in a sequence of strings with the behaviour's names.

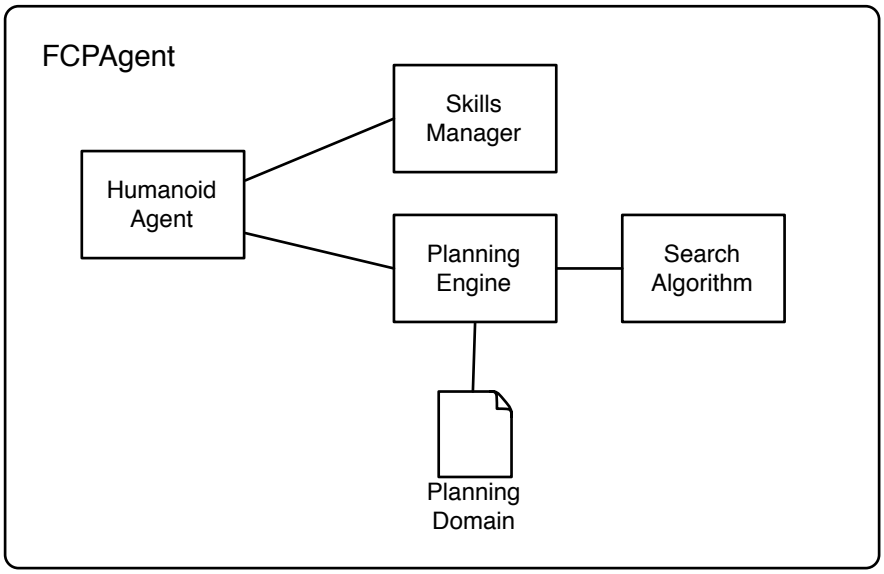


Figure 4.2: Architecture of the planning system.

4.3.2 Planning Domain Description

The planning domain is described in a configuration file, using the *libconfig* syntax. The *libconfig* library is used due to the simplicity of its parsing and it was already used in other parts of the agent, including in this dissertation (c.f. section 3.3.4).

In the description file, for each action it's required to specify:

- The behaviour name;
- A set of the behaviours that have to precede this behaviour (by default, all behaviours can precede);
- A set of preconditions that give permission to a behaviour to be executed. For instance, *HasTheBall* is a precondition for any Kick which states that the agent has to be very near to the ball. Other preconditions can be easily implemented.
- The duration of the behaviour's execution;
- The effect in the agent's position in both x and y axis (in meters);
- The effect in the agent's orientation (in degrees);

- The effect in the ball's position (in meters), which by default is $(x = 0, y = 0)$.

An extract of a domain description file is presented in listing 4.1. The sequence of the parameters is *ActionName*, *PreviousActions*, *Preconditions*, *Time*, *DeltaX*, *DeltaY*, *DeltaT*, *DeltaBallX* (0 by default) and *DeltaBallY* (0 by default). For the *PreviousActions* parameter, when given an empty list, all the actions are assumed.

Listing 4.1: Specification file for domain I.

```
ActionsDescription =
(
/*(ActionName,  PreviousActions, Preconditions, Time, DeltaX, DeltaY,
    DeltaT, DeltaBallX=0, DeltaBallY=0),*/
  ("Front", [], [], 1.08, 0.1, 0.0, 0.0),
  ("FrontF", ["Front", "FrontF"],["AlignedWBall"], 1.79, 0.28, 0.0, 0.0),
);
```

4.3.3 Sequence Behaviours

As it was said in 4.3.1, the result planning is given by a sequence of strings with the behaviours' names. To execute the sequence, we could just get the respective behaviours from the *Skills Manager* and execute them one by one until they were finished. However, sometimes the agent falls or the state changes from the one that was predicted in the planning, and the sequence has to be changed.

So, to make the code more understandable, these higher-level concerns were separated from the execution of the planning (which is lower-level), a new kind of behaviours was created: the *Sequence Behaviours*.

The *Sequence Behaviours* can be executed like any other behaviour: it has the *init*, *execute* and *finished*, *getName* routines. Besides, it receives a sequence of behaviours' names and executes it, providing also a *stop* method which allows to finish the behaviour in the middle of the sequence, but it waits for the current behaviour in the sequence to finished. This avoids the behaviour to be stopped in a non-stable agent's pose.

4.3.4 Implemented Methods

The Greedy and A* search algorithms were implemented. Further information about how these algorithms work was presented in the section 2.5.2. The search method is very similar in both methods. The only difference is at the evaluation function: the Greedy best-first algorithm only evaluates a node by its heuristic function while the A* algorithm uses the heuristic and the cost functions.

A variation used in the search methods is the limitation of the number of search steps. As the environment is very dynamic it's not desirable to have the agent lasting too much

time planning. It would probability be outdated when it would start executing the calculated plan. So, whenever it reaches a certain number of search steps, it returns the plan whose final state is the nearest to the goal state, among all the expanded search nodes.

4.3.5 Evaluation functions

The evaluation function was briefly described in section 4.2.3. However, mixing time, distances and orientations leads to a problem: different units of measurement. If we simply add the difference of the orientation in degrees and the distance in meters, the result will be much more influenced by the degrees, which can be between 0^o and 180^o. The same happens with the duration which is measured in seconds.

Concerning this, all the information needs to be properly normalized and weighted:

- The cost function corresponds to the sum of all durations divided by the ,maximum expected time to do that. For instance, the maximum time for the agent to reach the ball is the distance between the agent and the ball divided by the worst speed of the available actions.

  The equations 4.2, 4.3 and 4.4 show how to calculate this values, in which *start* is the initial node and *goal* is the goal node. Note that this is also applied to the other cases referred in the section 4.2.3, including the differences of orientation, in which the minimum angular speed is used.

$$v = \frac{ds}{dt} \Leftrightarrow dt = \frac{ds}{v} \tag{4.2}$$

$$timeBetweenStates(n_A, n_B) = \frac{distance(n_A, n_B)}{minSpeed} \tag{4.3}$$

$$g'(x) = \frac{\sum_{i=1}^{k} duration(x)}{timeBetweenStates(start, goal)} \tag{4.4}$$

- The heuristic function is the distance from a state to the goal state. So this distance is given in meters and degrees. Calculating the values of the distances and angles in the initial state, we have enough data to normalize the evaluation parameters in the intermediate states. This is illustrated in figure 4.3, in which point **1** respects to the agent's initial position, **2** is the agent's intermediate position and **3** is the agent's final position. This is stated with the equations 4.5 and 4.6.

$$h_{distance}(x) = \frac{distance(x, goal)}{distance(start, goal)} \tag{4.5}$$

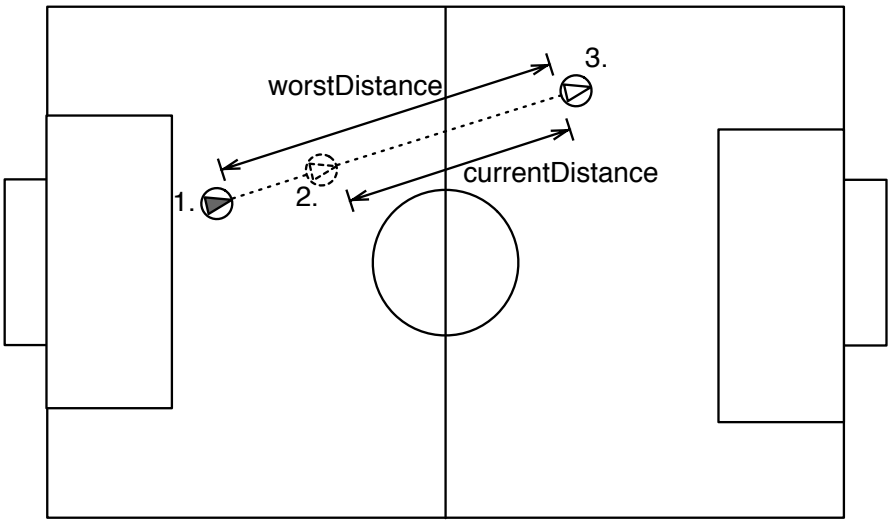


Figure 4.3: Normalization of the heuristic function.

$$h_{orientation}(x) = \frac{\Delta Orientation(x, goal)}{worstOrientation} \tag{4.6}$$

4.3.6 Solutions with Adaptive Resolution

Depending on the initial state, more or less precision is desirable on the resulting plan. For instance, if the initial state is very far from the goal state, the planner will have to make a bigger sequence, which makes the problem much more complex, and with that it will take longer to return the expected results. Besides, the environment is very dynamic and it's required that the agent is able to react very fast. By the other side, if the agent is very near, for example, to the ball, in order to execute a good kick, it's very important that the agent is in the right position, so the minimum error is required.

Thus, the acceptable error for the planning can be changed with regard to the distance between initial and goal states, and therefore the planner returns the desired results in real time.

4.3.7 Obstacle Avoidance

In a soccer game it is very important to be able to avoid obstacles. Many times, the players of the opponent team can be blocking the path to the ball or any other target.

This is implemented by penalizing actions in which the agent passes through such points. This is, a state which is near to an obstacle is more expensive.

The penalization of a state for an obstacle i is given by the following function:

$$g_i(x,y) = \begin{cases} \frac{-d_i(x,y)^2+r^2}{r^2} & \text{if } d_i(x,y) \leq r \\ 0 & \text{if } d_i(x,y) > r \end{cases} \tag{4.7}$$

The function receives the agent's position, x and y. The $d_i(x,y)$ function gives the distance from the agent to the obstacle i and r is the maximum distance in which the agent is penalized for being near to the obstacle – the *obstacle radius*, which was defined as $r = 1.0$.

This turns the obstacle avoidance to be smooth. This means that if the agent is in the circumference with centre in the obstacle and radius r, the penalization will be 0. However, as we decrease the circumference radius, the penalization increases as an hyperbolic line, reaching its maximum ($max(g_i(x,y)) = 1.0$) at 0. This can be seen in the graphs of figure 4.4.

So, the total penalization is given by the sum of the penalizations for all n obstacles:

$$g_{Obstacles}(x,y) = \sum_{i=0}^{n} g_i(x,y) \tag{4.8}$$

With this, to define the planning problem, in addition to the initial and goal states, also the set of obstacles can be specified.

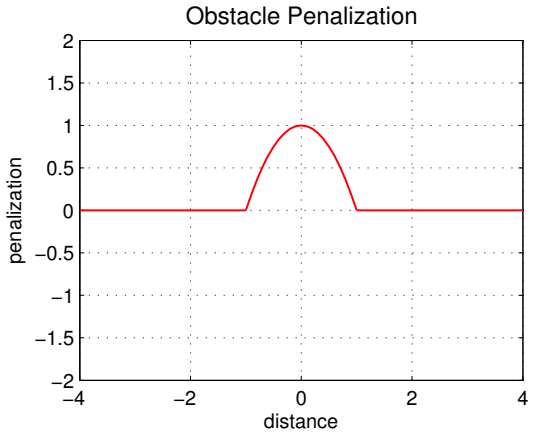
4.3.8 Replanning decision

An important aspect about the planning is when to re-plan. In a soccer game, the ball is always moving, as well as the players, and so on. Thus, the agent's will need to constantly change its goals. For this, the simplest solution was implemented: the agent is always re-planning.

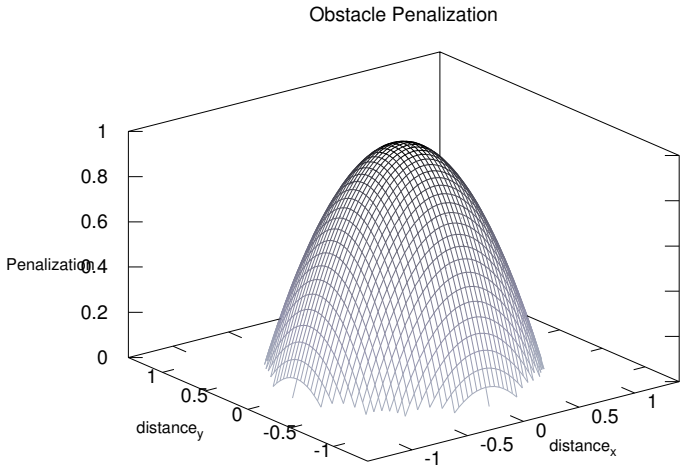
With this, the differences between the planned state and the real state are always being reviewed and also moving obstacles can be modelled, for instance opponent agents.

4.4 Experimental Results

The experiments made to prove the concept consist in putting the agent and the ball in specific positions, and then plant some obstacles in well-known positions. The goal state for the planning is the agent in a position 0.5 meters behind the ball and oriented to the opponent goal.



(a) Function of penalization over agent's distance to the obstacle.



(b) Function of penalization over agent's position relative to the obstacle's position.

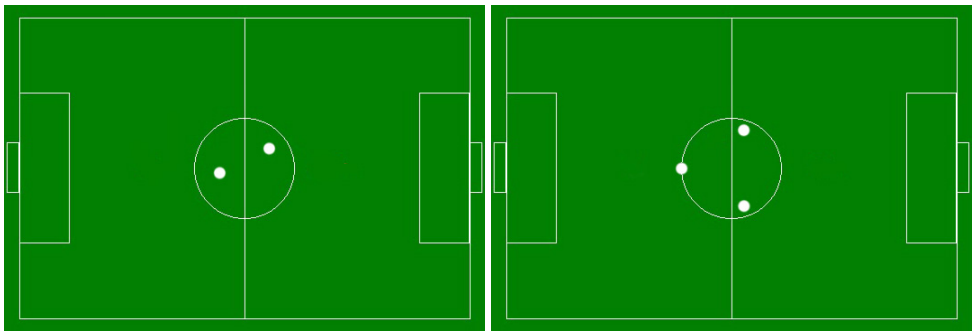
Figure 4.4: Graphs explaining how the obstacle penalization is calculated. This penalization is then added to the cost function for a given state.

Table 4.1: Domain I with a simple set of actions.

Action	Previous Actions	Time	DeltaX	DeltaY	DeltaT	DeltaBallX	DeltaBallY
"Front"	any	1.08	0.1	0.0	0.0	0.0	0.0
"FrontF"	"Front", "FrontF"	1.79	0.28	0.0	0.0	0.0	0.0
"RotLeft"	"Front", "RotLeft", "RotRight"	0.26	0.0	0.0	0.16	0.0	0.0
"RotRight"	"Front", "RotLeft", "RotRight"	0.26	0.0	0.0	-0.16	0.0	0.0

4.4.1 Setup

In this experiments, 2 different obstacles distributions were used, namely setups A and B, as illustrated in figure 4.5. The setup A, has obstacles in the points $(x = 1.0, y = -0.8)$ and $(x = -1.0, y = 0.2)$. The setup B, has obstacles in the points $(x = 0.5, y = 1.5)$, $(x = 0.5, y = -1.5)$ and $(x = -2.0, y = 0.0)$.



(a) A. (b) B.

Figure 4.5: Obstacles used for the experiments.

Also 2 different planning domains were used. The first, **actions I**, as presented in the table 4.1 has only a simple set of actions, while the second, **actions II**, presented in the table 4.2 has a wider set of actions.

Table 4.2: Actions in domain II.

Action	Previous Actions	Time	DeltaX	DeltaY	DeltaT	DeltaBallX	DeltaBallY
"FrontS"	any	0.44	0.025	0.0	0.0	0.0	0.0
"Front"	any	1.08	0.1	0.0	0.0	0.0	0.0
"FrontF"	"Front", "FrontF"	1.79	0.28	0.0	0.0	0.0	0.0
"Back"	any	1.0	0.08	0.0	0.0	0.0	0.0
"SideWalkRight"	any	0.9	0.0	0.12	0.0	0.0	0.0
"SideWalkLeft"	any	0.9	0.0	0.12	0.0	0.0	0.0
"RotLeft"	"Front", "RotLeft", "RotRight"	0.26	0.0	0.0	16.0	0.0	0.0
"RotRight"	"Front", "RotLeft", "RotRight"	0.26	0.0	0.0	-16.0	0.0	0.0

Table 4.3: Experimented planning problems.

Experiment	Obstacles	Actions	Agent's Initial Position	Agent's Final Position
1	None	II	$(x = -3.8, y = 0.0)$	$(x = -1.0, y = 1.0)$
2	A	I	$(x = -3.8, y = 0.0)$	$(x = 3.5, y = 0.0)$
3	B	I	$(x = -3.8, y = 0.0)$	$(x = 3.5, y = 0.0)$
4	A	II	$(x = -3.8, y = 0.0)$	$(x = 3.5, y = 0.0)$
5	B	II	$(x = -3.8, y = 0.0)$	$(x = 3.5, y = 0.0)$

Combining the 2 obstacles setups (A and B) with the 2 sets of actions (I and II) we get 4 possible situations. Table 4.3 lists the different experiments made, including this situations and a simple experiment without obstacles.

4.4.1.1 TacticalMap

An useful utility used in this work was the *TacticalMap*. The TacticalMap was previously developed Cruz and Cunha with the purpose of giving a more clear information about the agent's beliefs during a game, for instance where the agent believes to be the ball or even were the agent believes to be the ball after a few seconds.

It receives commands from an agent allowing to draw shapes or text information in a 2D soccer field in real time. [RRL11]

It was developed specially for the FCPortugal team and can be easily used with SimSpark and the FCPortugal agent.

In this experiment it was used to draw the obstacles, the initial position, the goal state and the agent with its position and orientation. In the results of section 4.4.2 some figures illustrating the experiments were taken using this utility.

4.4.2 Results

This section reports the results achieved with this planning approach.

In each experiment, was recorded the log file, generated by the SimSpark simulator, the output generated by the agent, a MATLAB®[2] log file, with the agent position and the server time, and a video capture of the screen, with the simulation and the TacticalMap graphical visualization.

4.4.2.1 Experiment 1

This experiment sets a very simple problem, in which there aren't any obstacles and the agent only needs to go from position $(x = -3.8, y = 0.0)$ to $(x = -1.0, y = 1.0)$.

The resulting sequence of actions was the following:

[2]MATLAB® is a numerical computing environment and fourth-generation programming language developed by The MathWorks.

1. RotRight
2. Front
3. Front
4. FrontF
5. FrontF
6. FrontF
7. Front
8. RotRight
9. Front
10. FrontF
11. FrontF
12. Front
13. Front
14. RotLeft
15. RotLeft
16. RotLeft
17. RotLeft
18. SideWalkLeft
19. SideWalkLeft
20. RotRight
21. SideWalkRight
22. SideWalkRight
23. RotRight
24. SideWalkLeft

With this sequence, the agent followed the trajectory presented in the figure 4.6, build upon the generated MATLAB® logs. It is noticeable that the trajectory is not smooth. This is because the position perceived by the agent has some noise, but for the purpose of the experiment it is acceptable. Note that the axis in the graph are slightly different from the SimSpark field: figure 4.7 shows that the y-axis in the SimSpark field is upside down.

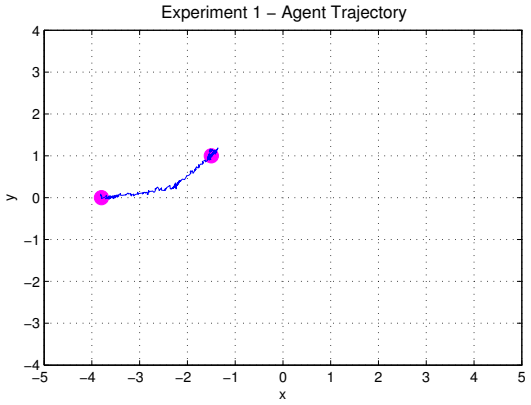


Figure 4.6: Trajectory of the agent during experiment 1. The initial and goal states are represented by the magenta circles.

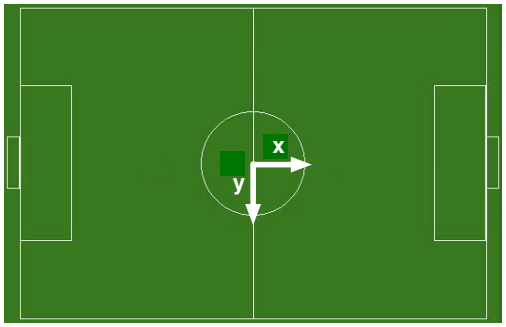


Figure 4.7: 2D axis of the simspark field.

Figure 4.8 presents a sequence of images illustrating the most interesting points of the experiment. The TacticalMap is in the top of each frame and the simulator is right below. In the bottom of each frame is stated the corresponding time. The first frame corresponds to the agent in its initial state and the last frame corresponds to the goal state.

The execution lasted 16.2 seconds.

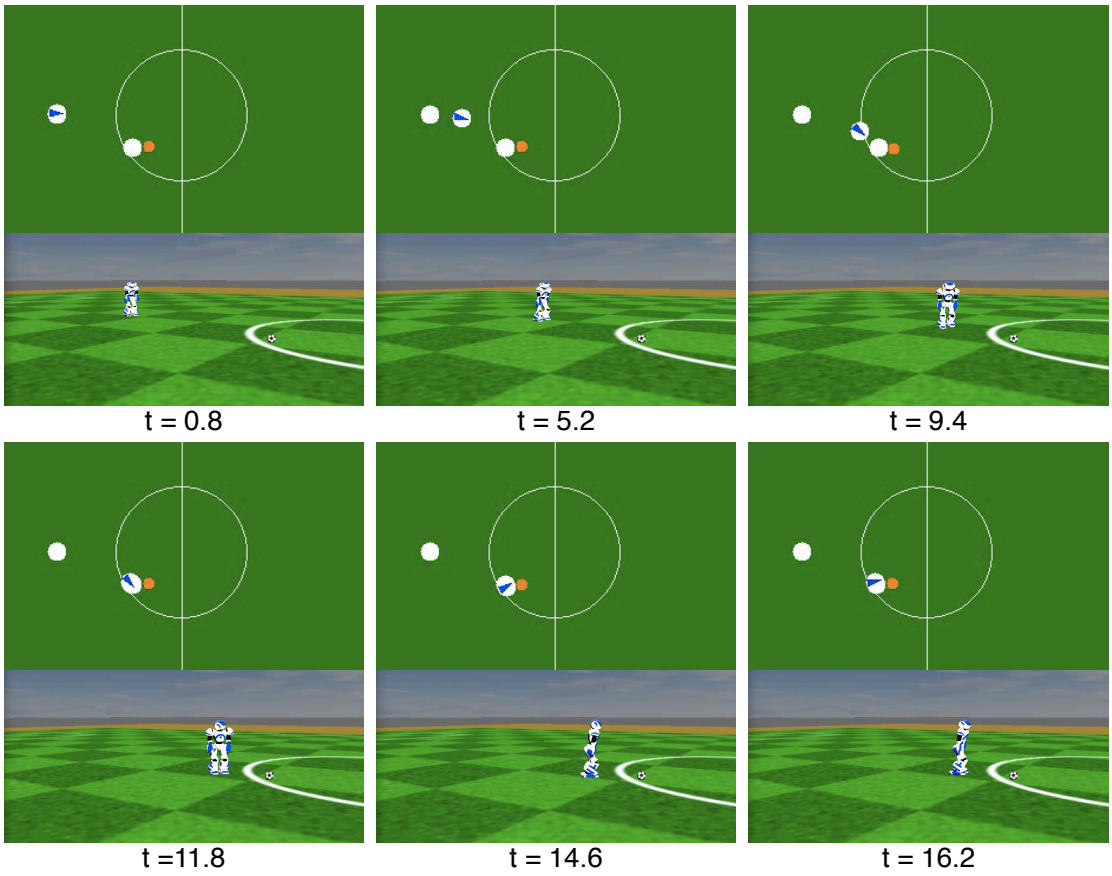


Figure 4.8: Image sequence of the video recorded during the experiment 1, showing the simulator and the *TacticalMap*.

4.4.2.2 Experiment 2

In this experiment, the agent is only capable of moving forward and rotating around itself (domain I).

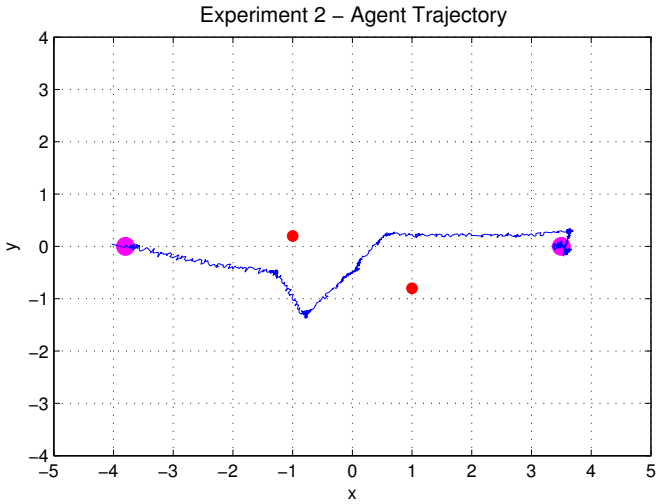


Figure 4.9: Trajectory of the agent during experiment 2. The red circles represent the obstacles and the initial and goal states are represented by the magenta circles.

Figure 4.9 shows the trajectory performed in this experiment.

The obtained image sequence is presented in figure 4.10, showing the most interesting points of the experiment. The TacticalMap is in the top of each frame and the simulator is right below. In the bottom of each frame is stated the corresponding time. The first frame corresponds to the agent in its initial state and the last frame corresponds to the goal state.

The agent took 57.2 seconds to execute the plan.

The agent goes from the point $(x = -3.8, y = 0.0)$ to the $(x = 3.5, y = 0.0)$, both marked with magenta circles. It executes the forward walking until it reaches a position near to the first obstacle. Although this is a simulation environment, there are also errors, and the agent didn't make a strictly forward walk, going slightly to the right. Then, after moving away from the obstacle the agent passes between the obstacles and then goes directly to the goal position.

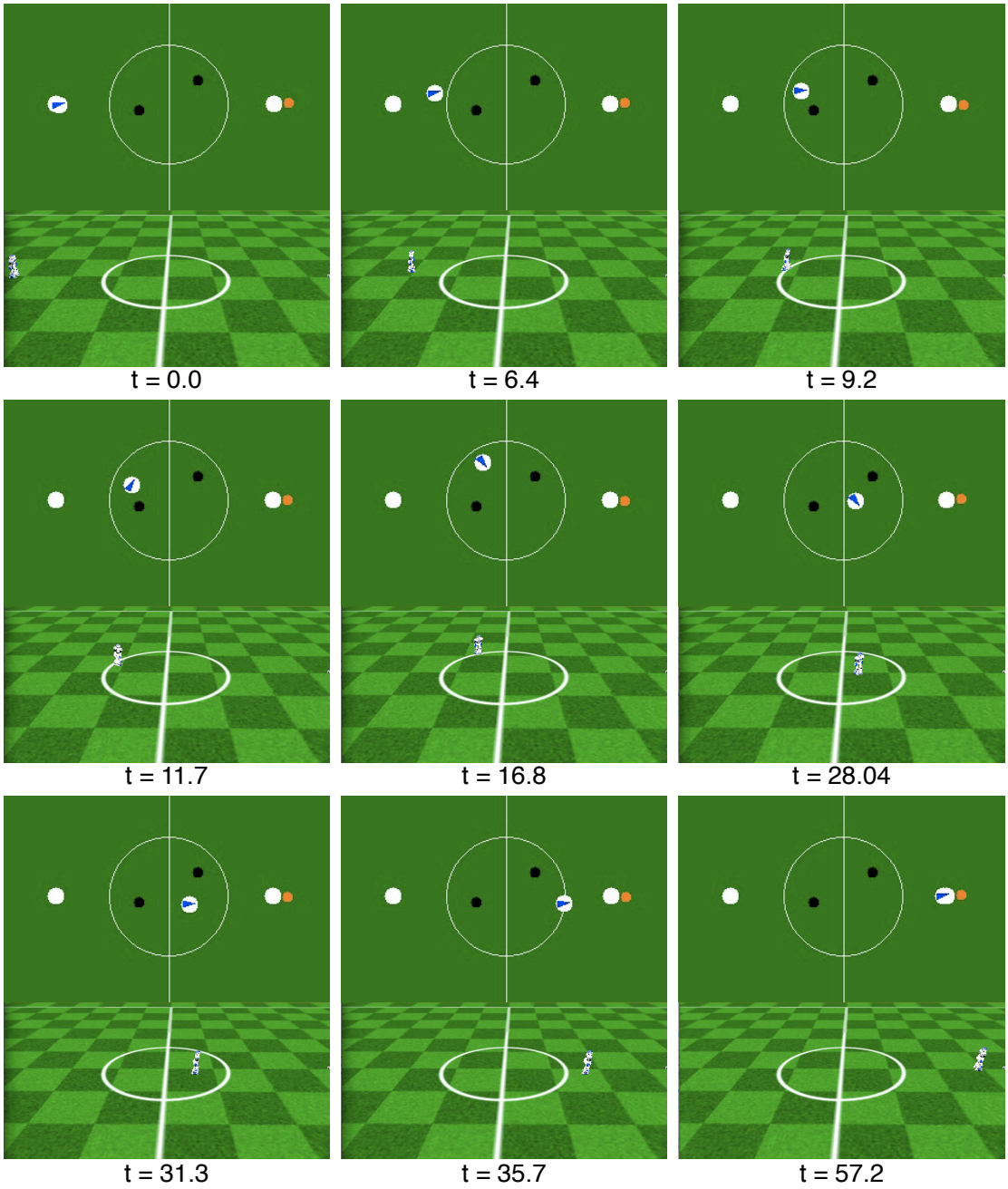


Figure 4.10: Image sequence of the video recorded during the experiment 2, showing the simulator and the *TacticalMap*.

4.4.2.3 Experiment 3

The same set of actions of the Experiment 3 was used in this experiment. However now we have a different distribution for the obstacles, the setup B, as described in the section 4.4.1.

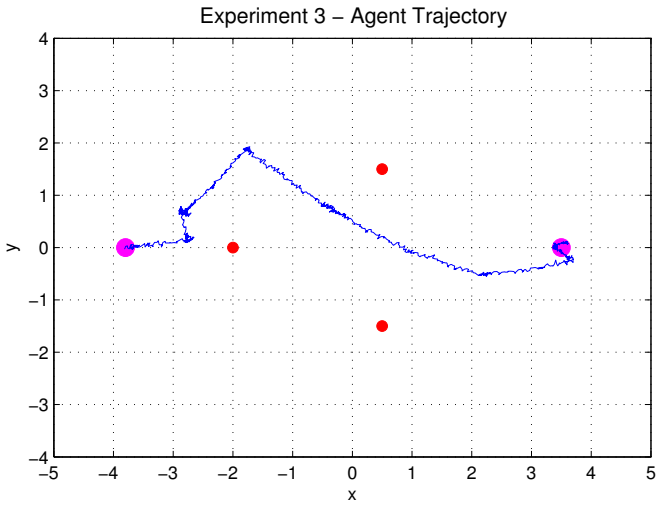


Figure 4.11: Trajectory of the agent during experiment 3. The red circles represent the obstacles an the initial and goal states are represented by the magenta circles.

The trajectory graph is shown in the figure 4.11. Again, the agent goes forward until it reaches an obstacle. Then it tries to go around the obstacle going to its left side. After that, it goes near to the goal position passing between two obstacles.

The image sequence of this experiment is in the figure 4.12. The agent took 57.3 seconds to reach the goal state.

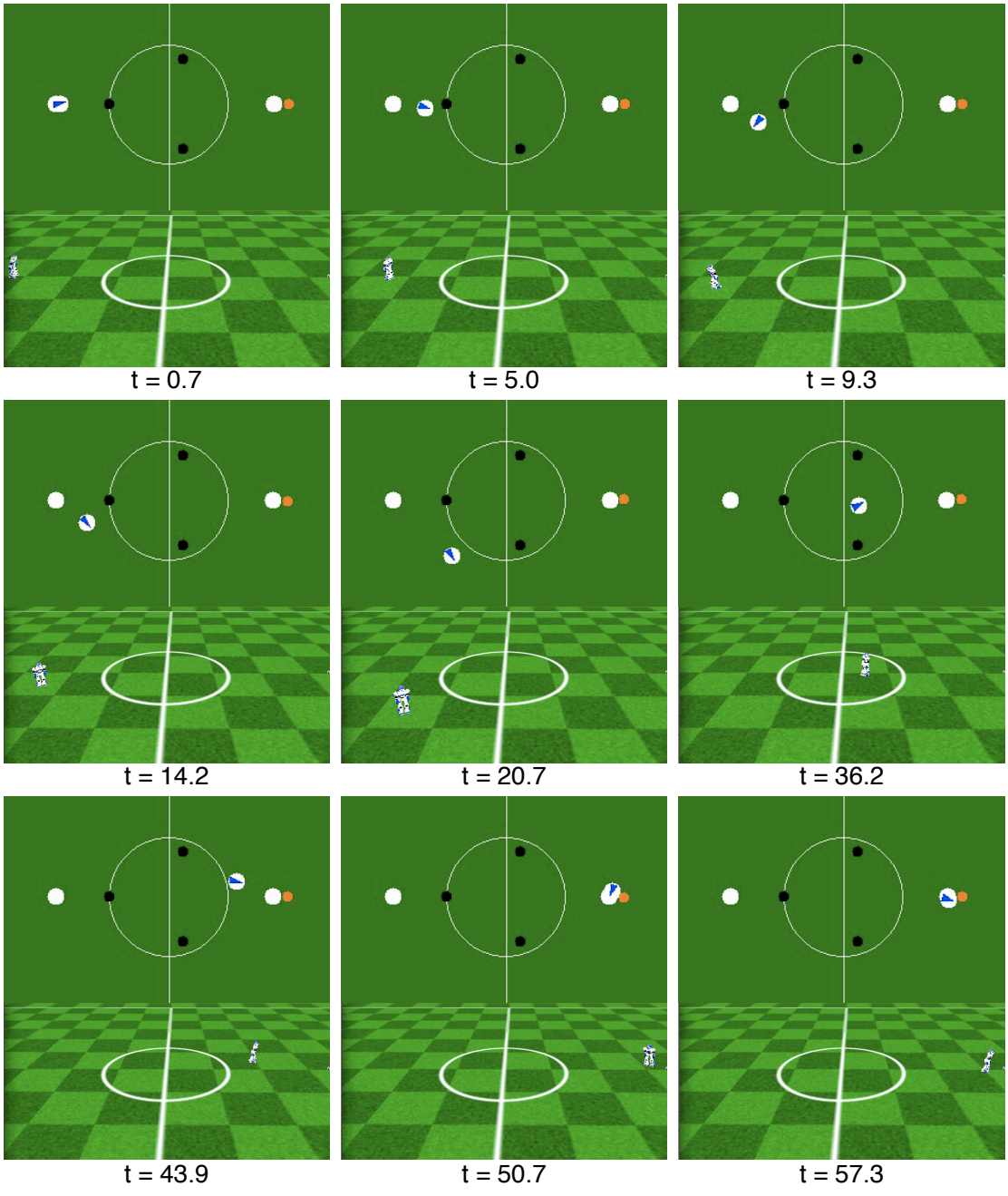


Figure 4.12: Image sequence of the video recorded during the experiment 3.

4.4.2.4 Experiment 4

The previous experiments had only a few set of actions, namely the *Front*, *FrontF*, *RotLeft* and *RotRight*. This time, a more complex set of actions is used. The actions *Back*, *SideWalkRight* and *SideWalkLeft* were added, which turns the planning to be more complex, but in the other hand to be more flexible to different situations.

Figure 4.13: Trajectory of the agent during experiment 4. The red circles represent the obstacles an the initial and goal states are represented by the magenta circles.

The trajectory graph is in the figure 4.13. Now, it's noticeable that the agent opted to do some side walk instead of simply rotate and go forward.

The time the agent took to reach the goal state was 52.1 seconds. As expected, this was less than the duration obtained in the experiment 2, in which the agent had the same obstacles setup (setup A) but few available actions.

The obtained image sequence is presented in the figure 4.14.

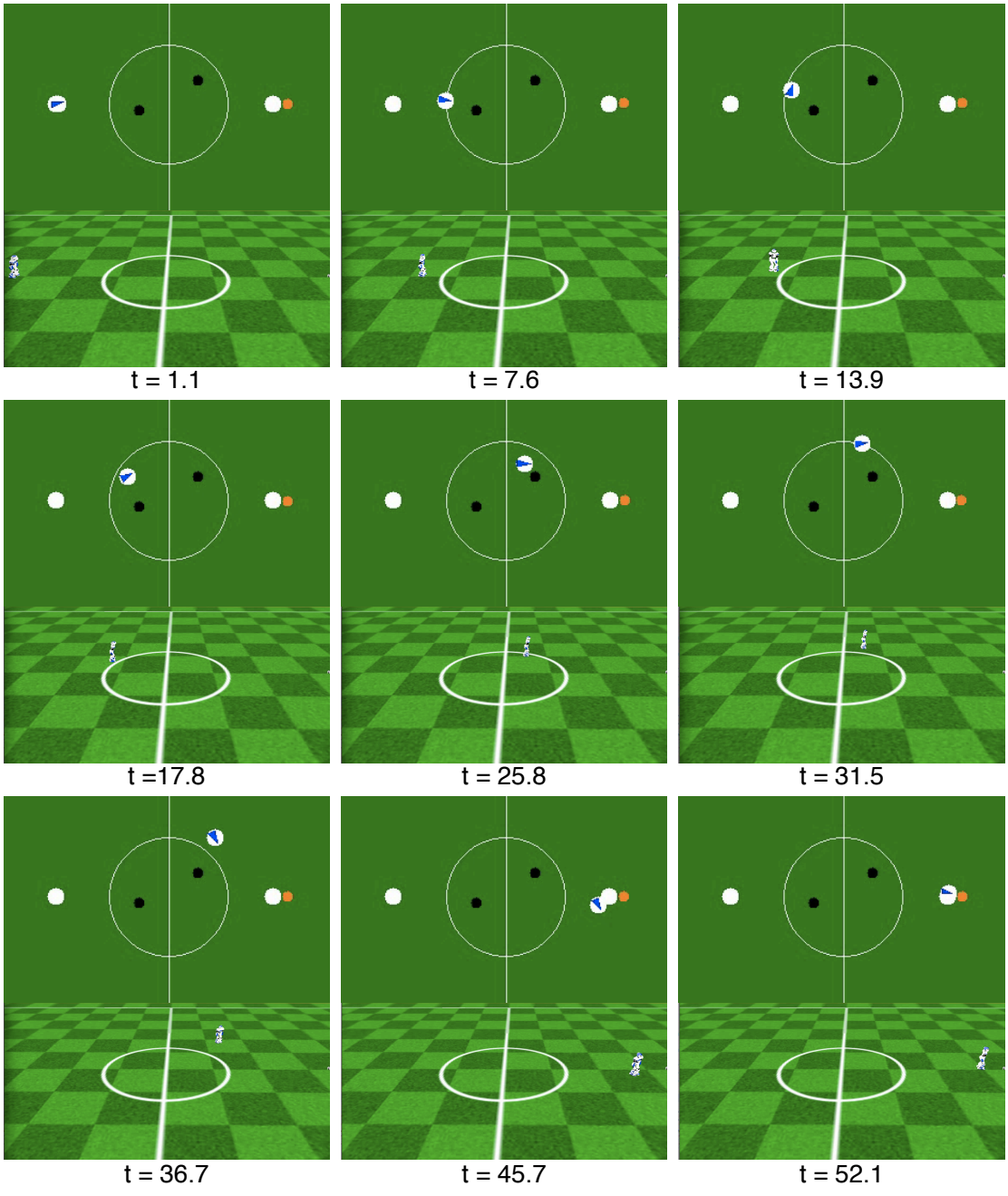


Figure 4.14: Image sequence of the video recorded in during the experiment 4.

Table 4.4: Duration of the execution of the plan for each experiment.

Experiment	Elapsed Time (sec)
1	16.2
2	57.2
3	57.3
4	52.1
5	56.9

4.4.2.5 Experiment 5

For the experiment 5 the same complex set of actions was used, but now with the obstacles setup B (c.f. figure 4.5).

Figure 4.15: Trajectory of the agent during experiment 5. The red circles represent the obstacles an the initial and goal states are represented by the magenta circles.

According to the trajectory graph of the experiment, shown in the figure 4.15, the agent executed a very mixed sequence of actions. This is not desirable due to the fact that switching between actions decreases the performance of the actions. Nevertheless, the experiment lasted 56.9 seconds, which was less than the obtained in the experiment 3, with the same setup of obstacles (setup B), but with a small set of actions.

The image sequence of the experiment is shown in the figure 4.16.

The duration of each planning executed by the agent in these experiments, is shown in the table 4.4.

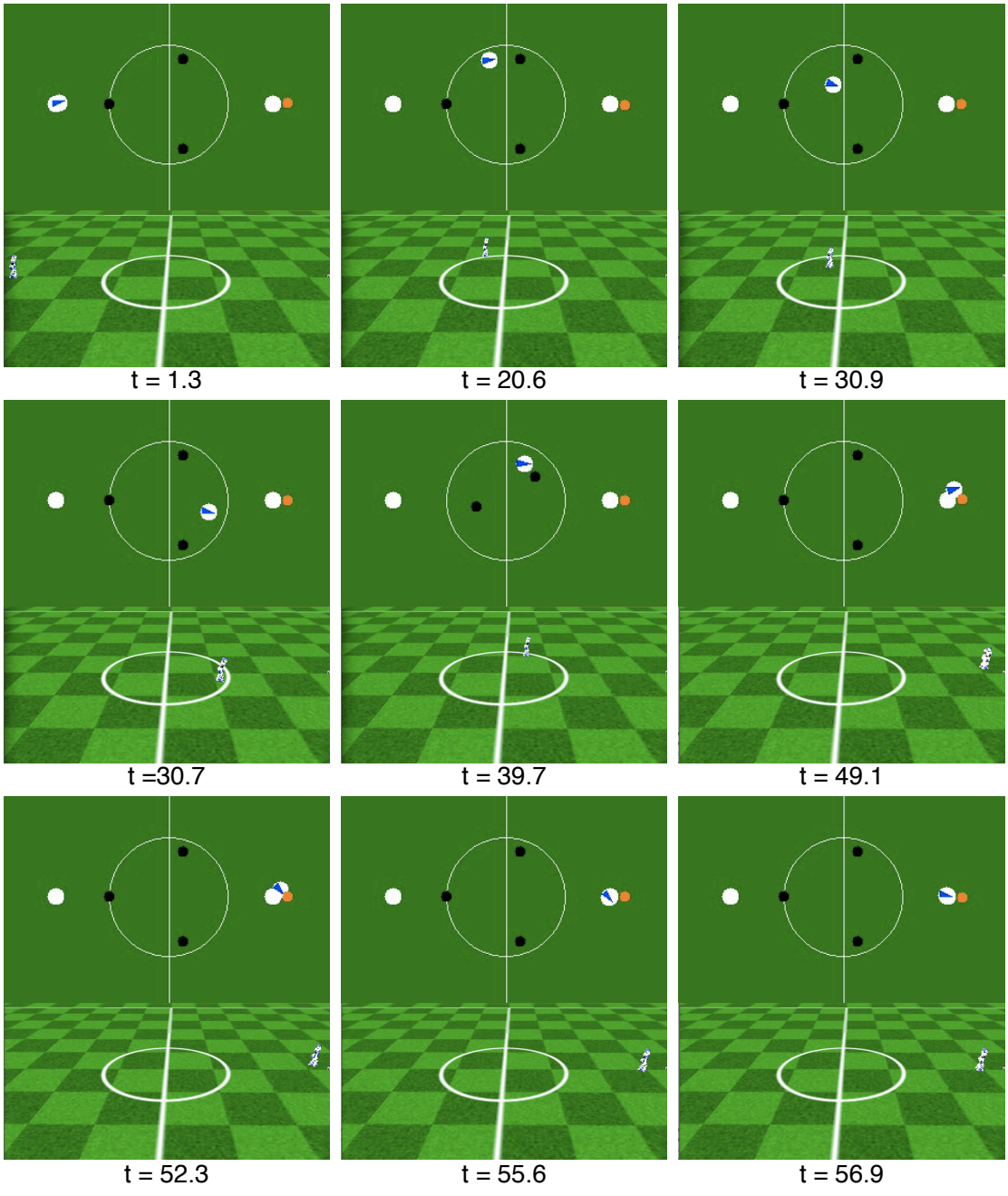


Figure 4.16: Image sequence of the video recorded during the experiment 5.

4.5 Summary

In this chapter, it was presented a planning approach to combine the different behaviours of the humanoid in a sequence that can perform a given task. The experiment results showed how the agent can move around the obstacles and the differences between the execution with different sets of actions and with different obstacles.

For future work, the planner could integrate in its set of actions the FCPortugal TFS walk [SKAJ09, SRL11]. The complexity of this stands on the fact that the TFS walk receives some dynamic parameters, like for instance the final position, which had to be discretized.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The work developed during this dissertation contributed to decrease the limitations of humanoids comparing to the humans. It was presented an exhaustive study of the work accomplished in this subject as well as the available technologies.

The FCPortugal team as a lot of research work developed on this field, which served as the base for this dissertation. So, the work was developed in a constructive manner, taking advantage from the work already developed, trying to improve it and fill the gaps.

In the optimization scope, some changes were made in the SimSpark simulator. It is the case of the features of changing the position of the ball or the humanoid in field and extending the server play time. The configuration file now allows to have a generic optimization that can be applied in other projects beyond the FCPortugal's agent.

It was created a side kick behaviour which was then optimized and with the Tabu Search method it reaches now 3.07 meters, which is an improvement of 116% in the distance, and from am initial duration of 1.14 seconds, it was possible to achieve 0.88 seconds, leading to an improvement of 30%.

The forward kick, which was previously used in the German Open 2011, was optimized reaching now 6.30 meters against the previous 5.15 meters which is an improvement of approximately 22% achieved with the Hill Climbing algorithm. In this particular case, the Tabu Search didn't lead to the best results because the forward kick is very sensitive to minor changes, and new solutions generated in the optimization process that were very similar to previous solutions were considered has being processed already, being discarded. However, this discarded solutions could lead to different results.

Also the problem of sequencing different actions was solved modelling it with a planning approach. This was challenging due to the real-time requirements and the different

available actions, but lead to some interesting results indeed. It was possible to obtain a system capable of sequencing the different available behaviours in an efficient manner and also capable of avoiding obstacles reaching a certain position and orientation with good precision.

5.2 Future Work

There is a lot of work and research that can be made based on the work developed during this dissertation.

For instance, in the optimization, the feedback given by the agent about the experiment could include more information about the stability of the execution. At this point, it's only possible to know whether the agent fell on the ground.

For instance in the planning, it would be great to have an automated way to classify the actions with their properties (*DeltaX*, *DeltaY*, and so on). It would simplify the process of describing the planning domain. Also the heuristic, instead of being a static method, could be a method implemented using a Machine Learning approach, which would improve the results as the agent gained experience.

Also, the achieved results could be even better if an omni-directional walk was integrated in the set of available actions. This kind of actions could be easily integrated with the developed planning system.

Another great feature could be integrating the planning system with the real NAO agent from the Portuguese Team [NLR+]. The Portuguese Team participates in the RoboCup SPL league and has been developing a NAO agent based on the FCPortugal's agent, trying to make a common agent for both real and simulated NAO. Thus, the planning system could be easily merged with only a few changes.

References

[AK88] E. Aarts and J. Korst. Simulated annealing and boltzmann machines. 1988.

[AVL85] E.H.L. Aarts and P.J.M. Van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips J. Res.*, 40(4):193–226, 1985.

[BA08] J. Boedecker and M. Asada. Simspark–concepts and application in the robocup 3d soccer simulation league. In *SIMPAR-2008 Workshop on The Universe of RoboCup Simulators, Venice, Italy*, 2008.

[BDR+10] Joschka Boedecker, Klaus Dorer, Markus Rollmann, Yuan Xu, Feng Xue, Marian Buchta, and Hedayat Vatankhah. *SimSpark User's Manual*, 1.2 edition, January 2010.

[Bek05] G.A. Bekey. *Autonomous robots: from biological inspiration to implementation and control*. The MIT Press, 2005.

[Cam02] J.L.E. Campos. Real-time well drilling monitoring using gocad. In *Proceedings of the 22nd Gocad Meeting*, volume 10. Citeseer, 2002.

[CdSSMM08] E. Colombini, A. da Silva Sim
"oes, A. Martins, and J. Matsuura. A framework for learning in humanoid simulated robots. *RoboCup 2007: Robot Soccer World Cup XI*, pages 345–352, 2008.

[Čer85] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[CFH+02] Mao Chen, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *RoboCup Soccer Server: Users Manual*. The RoboCup Federation, 2002.

[Cha11] Jason Chavis. The importance of robots. eHow, June 2011. URL: http://www.ehow.com/about_4596141_importance-robots.html.

[CIR99] CIRL. Planning and scheduling materials. The Computational Intelligence Research Laboratory of the University of Oregon, available at: http://www.cirl.uoregon.edu/research/overview.html, 1999.

[CP06] S. Carpin and E. Pagello. The challenge of motion planning for humanoid robots playing soccer. In *Proceedings of the Workshop on Humanoid Soccer Robots of the 2006 IEEE-RAS International Conference on Humanoid Robots*, pages 71–77, 2006.

[DA89] Anton Dekkers and Emile Aarts. Global optimization and simulted annealing. *Mathematical Programming*, 50(1-3):367–393, June 1989.

[DP85] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[FJL+88] S. Ford, J. Joseph, D. Langworthy, D. Lively, G. Pathak, E. Perez, R. Peterson, D. Sparacin, S. Thatte, D. Wells, et al. Zeitgeist: Database support for object-oriented programming. *Advances in Object-Oriented Database Systems*, pages 23–42, 1988.

[FL03] M. Fox and D. Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20(1):61–124, 2003.

[G+89] F. Glover et al. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[GEKO09] D. Grunberg, R. Ellenberg, Y. Kim, and P. Oh. Creating an autonomous dancing robot. In *Proceedings of the 2009 International Conference on Hybrid Information Technology*, pages 221–227. ACM, 2009.

[Gep04] L. Geppert. Qrio, the robot that could. *Spectrum, IEEE*, 41(5):34–37, May 2004.

[GHB+08] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. The nao humanoid: a combination of performance and affordability. *Arxiv preprint*, 2008.

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. *PDDL — The Planning Domain Definition Language*. AIPS-98 Planning Competition Committee, 1.2 edition, October 1998.

[GHP+97] Perry Gray, William Hart, Laura Painton, Cindy Philips, Mike Trahan, and John Wagner. A survey of global optimization methods. URL: http://www.cs.sandia.gov/opt/survey/main.html, Sandia National Laboratories, March 1997.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[Glo90] F. Glover. Tabu search-part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[Gui04] Guiness. *Guinness World Records 2005*. Guiness, 50th annv edition, August 2004.

[GZLR+05] K. Ghazi-Zahedi, T. Laue, T. Röfer, P. Schöll, K. Spiess, A. Twickel, and S. Wischmann. Rosiml-robot simulation markup language. *Retrieved at*, 20(05.2007), 2005.

[Hag03] C. Hagis. History of robots. 2003.

[Hes02] Arik Hesseldahl. Ten o'clock tech - say hello to asimo. *Forbes.com*, 2002.

[HH04] Randy L. Haupt and Sue Ellen Haupt. *Pratical Genetic Algorithms*. Wiley - Interscience, second edition, 2004.

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[Hol75] J.H. Holland. Adaptation in natural and artificial systems. 1975.

[Hon07] Honda. Asimo technical information, September 2007.

[IdFC07] R. Ierusalimschy, L.H. de Figueiredo, and W. Celes. The evolution of lua. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 2–1. ACM, 2007.

[IDFF96] R. Ierusalimschy, L.H. De Figueiredo, and W.C. Filho. Lua-an extensible extension language. *Software Practice and Experience*, 26(6):635–652, 1996.

[Jac07] J. Jackson. Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82–87, dec. 2007.

[JM03] O.C. Jenkins and M.J. Mataric. Automated derivation of behavior vocabularies for autonomous humanoid motion. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 225–232. ACM, 2003.

[KAK+97] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for ai. *AI magazine*, 18(1):73, 1997.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671, 1983.

[KHK+08] K. Kaneko, K. Harada, F. Kanehiro, G. Miyamori, and K. Akachi. Humanoid robot hrp-3. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2471–2478. IEEE, 2008.

[LR07] Nuno Lau and Luis Paulo Reis. *FC Portugal - High-level Coordination Methodologies in Soccer Robotics*, pages 167–192. Itech Education and Publishing, Vienna, Austria, December 2007.

[LR08] Tim Laue and Thomas Röfer. Simrobot - development and applications. In Workshop Proceedings of SIMPAR 2008, November 2008.

[LSR06] T. Laue, K. Spiess, and T. R
"ofer. Simrobot–a general physical robot simulator and its application in robocup. *RoboCup 2005: Robot Soccer World Cup IX*, pages 173–183, 2006.

[MCDB+10] GA Medrano-Cerda, H. Dallali, M. Brown, NG Tsagarakis, and DG Caldwell. Modelling and simulation of the locomotion of humanoid robots. In *UK Automatic Control Conference, Coventry, 7-10 September*, 2010.

[MGH+98] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. *The AIPS-98 Planning Competition Comitee*, 1998.

[Mic98] O. Michel. Webots: Symbiosis between virtual and real mobile robots. In *Virtual Worlds*, pages 254–263. Springer, 1998.

[Mit98] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, February 1998.

[Mou11] C. Mouton. A study of the existing libraries to read from configuration files (from c++). *Arxiv preprint arXiv:1103.3021*, 2011.

[MSV+08] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori. The icub humanoid robot: an open platform for research in embodied cognition (special session on eu-projects). In *PerMIS '08 Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, pages 50–56, 2008.

[new] New castle university - engineering design centre's website, consulted in june, 2011. URL: http://www.edc.ncl.ac.uk/.

[NKSD11] C. Nikolopoulos, D. Kuester, M. Sheehan, and S. Dhanya. Investigation on requirements of robotic platforms to teach social skills to individuals with autism. *Human-Robot Personal Relationships*, pages 65–73, 2011.

[NLR+] AJR Neves, N. Lau, L.P. Reis, A.P. Moreira, A. Trifan, B. Pimentel, C. Sobrinho, and E. Domingues. Spl portuguese team: Team description paper for robocup 2011.

[OTI+00] K. Ogawara, J. Takamatsu, S. Iba, T. Tanuki, H. Kimura, and K. Ikeuchi. Acquiring hand-action models in task and behavior levels by a learning robot through observing human demonstrations. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*. Citeseer, 2000.

[oxf] World english oxford dictionary online, consulted in june 2011. URL: http://oxforddictionaries.com.

[PGL+09] Hugo Picado, Marcos Gestal, Nuno Lau, Luís Paulo Reis, and Ana Maria Tomé. Automatic generation of biped walk behavior using genetic algorithms. In Joan Cabestany, Francisco Sandoval Hernández, Alberto Prieto, and Juan M. Corchado, editors, *IWANN (1)*, volume 5517 of *Lecture Notes in Computer Science*, pages 805–812. Springer, 2009.

[Pic08] Hugo Picado. Development of behaviors for a simulated humanoid robot. Master's thesis, Universidade de Aveiro, 2008.

[Pin07] C.M.A. Pinto. Central pattern generator for legged locomotion: a mathematical approach. *ROBOMAT 07*, page 153, 2007.

[Rai08] R. Steven Rainwater. Ping pong playing robots. *Robots.net*, June 2008.

[Rei10] Luis Rei. Optimizing simulated humanoid robot skills. Master's thesis, FEUP - Faculdade de Engenharia da Universidade do Porto, 2010.

[RI06] L. Righetti and A.J. Ijspeert. Programmable central pattern generators: an application to biped locomotion control. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1585–1590. IEEE, May 2006.

[RL01] L. P. Reis and N. Lau. Fc portugal team description: Robocup 2000 simulation league champion. *RoboCup 2000: Robot Soccer World Cup IV*, pages 29–40, 2001.

[RLB+08] T. Röfer, T. Laue, A. Burchardt, E. Damrose, J. Müller, and A. Rieskamp. B-human team description for robocup 2008. 2008.

[RLM+] T. Röfer, T. Laue, J. Müller, A. Fabisch, K. Gillmann, C. Graf, A. Härtl, A. Humann, and F. Wenk. Team description for robocup 2011.

[RLM+09] T. Röfer, T. Laue, K. Müller, O. Bösche, A. Burchardt, E. Damrose, K. Gillmann, C. Graf, T. J. Haas, A. Härtl, A. Rieskamp, A. Schreck, I. Sieverdingbeck, and J. Worch. B-human team report and code release 2009. November 2009.

[RLML10] Luis Paulo Reis, R. Lopes, Luis Mota, and Nuno Lau. Playmaker: Graphical definition of formations and setplays. In A. Rocha, C. F. Sexto, L. P. Reis, and M. P. Cota, editors, *Proceedings of the 5th Iberian Conference on Information Systems and Technologies, CISTI 2010*, pages 582–587, Santiago de Compostela, Spain, June 2010.

[RLO01] L. P. Reis, N. Lau, and E. Oliveira. Situation based strategic positioning for coordinating a team of homogeneous agents. *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, pages 175–197, 2001.

[RLPA08] Luis Paulo Reis, Nuno Lau, Hugo Picado, and Nuno Almeida. Paper contributions, results and work for the simulation community of fcportugal. *Proceedings CD of RoboCup 2008*, 2008.

[RN10] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition, 2010.

[rob] History of robots. RobotWorx - Used Robot Division's website, consulted in June, 2011. URL: http://www.used-robots.com/robot-education.php?page=robot+history.

[RRL11] Luis Rei, Luis Paulo Reis, and Nuno Lau. Optimizing a humanoid robot skill. *Proceedings of the 11th International Conference on Mobile Robots and Competitions*, pages 84–89, 2011.

[S+05] R. Smith et al. Open dynamics engine, 2005.

[SHE03] T. SHEERAN. History of robots. 2003.

[SKAJ09] N. Shafii, A. Khorsandian, A. Abdolmaleki, and B. Jozi. An optimized gait generator based on fourier series towards fast and robust biped locomotion involving arms swing. In *Automation and Logistics, 2009. ICAL'09. IEEE International Conference on*, pages 2018–2023. IEEE, 2009.

[SRL11] N. Shafii, L. Reis, and N. Lau. Biped walking using coronal and sagittal movements based on truncated fourier series. *RoboCup 2010: Robot Soccer World Cup XIV*, pages 324–335, 2011.

[SWA+02] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: System overview and integration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483. IEEE, 2002.

[UMK+11] Daniel Urieli, Patrick MacAlpine, Shivaram Kalyanakrishnan, Yinon Bentor, and Peter Stone. On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In Kagan Tumer, Pinar Yolum, Liz Sonenberg, and Peter Stone, editors, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, volume 2, pages 769–776. IFAAMAS, May 2011.

[VAL94] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by local search. In *INFORMS Journal on Computing*, pages 91–120. Citeseer, 1994.

[Wei09] Thomas Weise. Global optimization algorithms – theory and application. *URL: http://www.it-weise.de, Abrufdatum*, 2009.

[WF] Derek Wadsworth and Doug Few. Idaho national laboratory's website - humanoid robotics, cunsulted in may, 2011. URL: https://inlportal.inl.gov/portal/server.pt/community/robotics_and_intelligence_systems/455.

[WKoT99] R.A. Wilson, F.C. Keil, and Massachusetts Institute of Technology. *The MIT encyclopedia of the cognitive sciences*. MIT Press, 1999.

[WW04] M. Wetter and J. Wright. A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization. *Building and Environment*, 39(8):989–999, August 2004.