Optimization Approach for the Development of Humanoid Robots' Behaviors

Luis Cruz[1,5], Luis Paulo Reis[1,4], Nuno Lau[2,6], Armando Sousa[3,5]

[1]LIACC - Artificial Intelligence and Computer Science Lab., Univ. of Porto, Portugal
[2]IEETA - Institute of Electronics and Telematics Engineering of Aveiro, Portugal
[3]INESC-TEC - Institute for Systems and Computer Engineering of Porto, Portugal
[4]DSI/EEUM - Inf. Systems Dep., School of Engineering, Univ. Minho, Portugal
[5]FEUP - Faculty of Engineering of the University of Porto, Porto, Portugal
[6]DETI/UA - Electronics, Telecomm. and Informatics Dep., Univ. Aveiro, Portugal
ei06034@fe.up.pt, lpreis@dsi.uminho.pt, nuno.lau@ua.pt, asousa@fe.up.pt

Abstract. Humanoid robots usually have a large number of degrees of freedom which turns humanoid control into a very complex problem. Humanoids are used in several RoboCup soccer leagues. In this work, the SimSpark simulator of the Simulation 3D will be used. This paper presents an automatic approach for developing humanoid behaviors based on the development of a generic optimization system. The paper describes the adopted architecture, main design choices and the results achieved with the optimization of a side kick and a forward kick. For both skills, the optimization approach allowed the creation of faster and more powerful and stable behaviors.

Keywords: Humanoids, Soccer, Simulation3D, Optimization, NAO, SimSpark

1 Introduction

The advent of humanoid robots has brought new challenges in robotics and related fields. The development of behaviors with many degrees of freedom (DOF) is one of these challenges. Within RoboCup [7], an international robotic competition, researchers are challenged to many of these problems in order to have robots, including humanoids, playing soccer and performing other complex tasks.

The RoboCup Simulation 3D League uses a model of the humanoid robot NAO [4] and was created in order to promote research in techniques to make humanoid robots play soccer. The FCPortugal Team project was conceived as an effort to create intelligent players, capable of thinking like real soccer players and behave like a real soccer team competing in the RoboCup Simulation Leagues. The result of this kind of research may be extended to other domains, such as the use of real humanoid robots to perform social tasks such as helping the blind to cross streets or elderly people to perform tasks that became impossible for them to do alone.

The use of simulated environments is very useful because it allows the developers to make arbitrary or complex tests in the simulator without using the real robot thus avoiding expensive material getting damaged [8].

In order to have a humanoid robot playing soccer, it has to walk in many directions, kick the ball, get up when necessary, etc. However, developing a behavior is not trivial as you have to control lot of DOF (NAO humanoid has 22) in a complex body shape where most configurations are not stable [9][13]. This makes the development of fast and stable behaviors a complex task. This paper presents an automatic approach to improve the development of such behaviors using generic optimization methods.

This paper starts by describing the problem in Section 2, then the designed optimization system and its main features in Section 3. The results achieved are presented in Section 4 and Section 5 presents the conclusion.

2 Problem Formulation

The official Robocup 3D simulation server is SimSpark [2], a generic physical multi-agent simulator system for agents in 3D environments. It simulates the physics dynamics of the real world in the context of a soccer game as well as the robots (physical dimensions, sensors and joints).

2.1 Humanoid Robot Behaviors

A behavior or skill consists in a intentional, repeatable action executed by the agent, such as getting up from a fall or kicking the ball. These can be specified once and executed as many times as necessary. The FCPortugal's agent has these behaviors specified and stored in XML files, providing the parameters of a behavior specification model. All behaviors are loaded when the agent starts.

A behavior defines the trajectories for all the humanoid joints. A joint trajectory can be defined by a set of points over an interval of time. A behavior specification model can be used to specify skills by computing different joint trajectories. There are various models for specifying skills, some of those used by the FCPortugal Team, including the StepBehavior model which uses step functions to model the trajectories, SlotBehavior model which uses a Sine Interpolation (SI) method [11] [12] [13], and the Central Pattern Generator model (CPG), based on the work of Behnke [1].

3 The Optimizer

The optimizer was designed in a distributed architecture allowing a behavior to be optimized using either a single computer or multiple computers connected via a network. Figure 1 depicts the configuration of the optimizer.

Once started, the optimizer server starts the simulator, the monitor and the agents using predefined shell scripts. It may start multiple simulators and their respective monitors in different computers if specified. The agents connect to the simulation server and to the optimization server. Then the optimization server sends to the agents the behaviors to be executed and receives from them information about their execution used to compute the behavior performance.

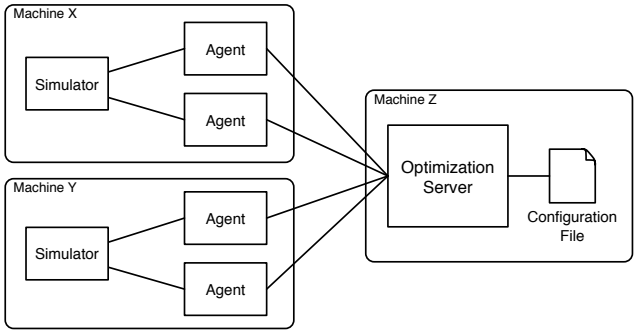


Fig. 1: Possible setup for the optimizer, running in multiple machines.

The optimization server is the core of the system and it is multi-threaded. There is one thread per agent that provides the agent with the behavior to optimize, waits for the agent to finish executing the behavior, receives the experimental data sent by the agent and evaluates it, giving that generated behavior a score according to a specified objective function.

A shell script is responsible of starting the FCPortugal agents. This allows the agents to be started on the local or in a remote machine. Another shell script terminates the agents. In the current implementation, for convenience, the simulator script, that starts the simulator, is started from the agent script.

The scripts exist for both flexibility and to isolate the optimization server from the details of running the other components. Recompiling the optimization server each time we wanted to change the parameters of the agents, such as when the host of the simulation server changes, would be a poor design choice [12].

3.1 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 how the agent will execute the experiment. The main parameters are the following:

- **behavior** - Path to the XML specification file of the behavior to optimize.
- **type** - Type of behavior. It can be "StepBehavior", "SlotBehavior", "ExpediteSlotBehavior" or "CPGBehavior".
- **objectiveScript** - Path to the LUA script that evaluates the executed experiment and returns its fitness.
- **objective** - The evaluation method can be implemented directly in the optimizer code, instead of in a LUA script. This is not recommended since most of the fitness functions can be specified with a LUA script. However, for a fitness function which handles more complex data this might be useful.
- **script** - Path to the script responsible for starting agent(s) and simulator(s).
- **name** - Name for the resulting optimized behavior.

- **algorithm** - Algorithm used for optimization. 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 optimization can be made in less time. This option specifies the number of agents that will be executing.
- **waitBefore** - Time in seconds that the agent will wait before executing the experiment. Ater a beam the agent loses some balance and this waiting time may be used to stabilize or to perform other preparation taks.
- **useGyroStabilization** - Instead of using the waitBefore option, another approach is possible. Using the data received from the agent's gyroscope, it is possible to know if the agent has enough stability to start the execution. The value in this parameter is a boolean (true/false).
- **waitAfter** - Time in seconds, the agent waits before collecting data about the experiment to be evaluated by the fitness function. This is useful, for instance when optimizing a kick as we need to know the final ball position.
- **beamBall** - The position of the ball in the start of each execution. Again, if when optimizing a kick we need to reset the ball position after each kick.
- **algorithm parameters** - the parameters of the algorithm such as the population size for genetic algorithms or the size of the tabu list for *Tabu Search*.
- **behavior optimization parameters** - specific parts of the behavior to be optimized such as which slots or steps for SlotBehaviors or StepBehaviors, respectively, which joints restrictions to enforce such as symmetries, etc.
- **minChange** - minimum value to change an optimization variable to obtain a neighboring solution - does not apply to time intervals
- **maxChange** - maximum value to change an optimization variable to obtain a neighboring solution - does not apply to time intervals
- **minChangeDelta** - minimum value to change an optimization variable that represents a time interval to obtain a neighboring solution;
- **maxChangeDelta** - maximum value to change an optimization variable that represents a time interval it to obtain a neighbouring solution.
- **serverRestartTime** - maximum amount of time the simulator can run without being restarted (in seconds). Further details in section 3.5.

To process the configuration files *libconfig*[1] library is used. For this kind of configuration it is more compact and readable than XML. Besides, it is type-aware, which means that it is not necessary to do string parsing to the specified values [10].

3.2 Gyroscope Initial Stabilization

The time the agent has to wait before starting the experiment can be statically fixed in the configuration file or, alternatively, this time can be dinamically

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

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

3.3 Objective Function

In order to evaluate the experiment, the agent collects 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 and final position; The distance travelled by the agent in each axis; The final agent's orientation; The ball's initial and final position; The distance travelled by the ball in each axis; The simulation time.

After receiving this data, the server evaluates the experiment. 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. Furthermore, it is necessary to recompile the optimizer. To handle this drawback, we used a scripting language known as Lua [5, 6]. 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 [3]. 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.

3.4 Optimization Algorithms

Several problems consist in finding the best configuration of a set of variable values to minimize (or maximize) some quantity known as the objective function, f. The independent variables that can change while searching for an optimum are known as decision variables. For some problems, the values that decision variables can take are specified by their domains and also through a number of conditions known as constraints.

The search space of a problem is defined as the set of all candidate solutions. A candidate solution is an instantiation of the decision variables, if it satisfies all the constraints of the problem it is called a feasible solution or just a solution. The solution space is the set of all (feasible) solutions. An optimal solution is a solution where the objective function evaluates to a minimum (or maximum). Computer algorithms that traverse the search space with the purpose of finding optimal solutions are called automatic optimization algorithms.

3.5 Adaptations made in the simulator

Whenever an experiment is made, the agent and ball positions change. In order to have the same conditions for every execution, it is necessary to undo these changes. It is necessary to have an automated way of controlling the agent and ball positions. So, a few changes were made in the simulation server.

The simulator has a feature that allows the creation of plugins. These plugins can be actions that make some changes in the simulation environment and they are triggered when the server receives some specific messages. The simulator had already a plugin that allowed the repositioning of the agent – the *Beam* plugin. However, this could only be done in the beginning of the games. As the optimizer needs to do this in every game states, a few changes were made in the *Beam* plugin. To reposition the ball, a similar plugin was created.

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

4 Experimental Results

4.1 Side Kick Optimization

A good approach for developing new behaviors for the FCPortugal agent, is to start by creating a version of the behavior by hand. After that, the optimizer can be used to enhance it. In this experience, a side kick was developed using the Slot Behaviour model. The execution of the non optimized behaviour can be seen in figure 2. 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.

$$f(x) = \frac{bdist_y^3 * bell(bdist_x)}{\Delta t} \tag{1}$$

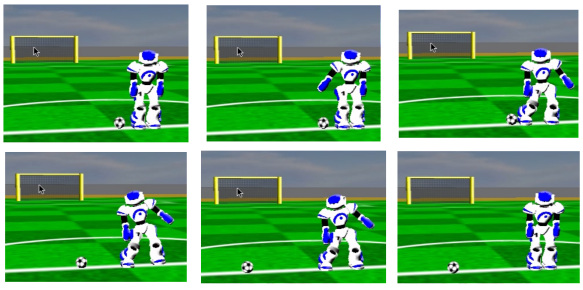


Fig. 2: Sequence of images showing the execution of the side kick skill.

Table 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 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

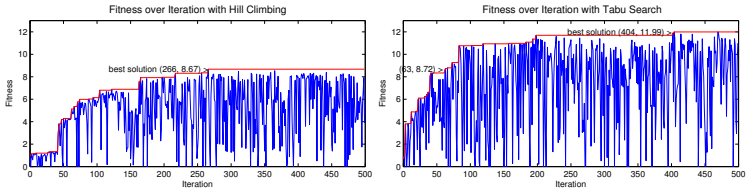
The function $bell(x)$ is the probability density function of a normal (or Gaussian) distribution. It is used in the fitness function with $\mu = 0.0$ (a.k.a. *mean*) and $\sigma^2 = 1.0$ (a.k.a. *variance*) with the aim of giving better fitness values to solutions in which the ball goes strictly to the side.

The optimization parameters used are shown in Table 1. The results achieved can be seen in Table 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 graphs of Figure 3 present the evolution of the fitness score of the best and current solutions, for both Hill Climbing and Tabu Search methods. The blue lines represent the current experiment fitness and the red lines are the best fitness until that iteration. We can notice that 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.

4.2 Forward Kick Optimization

Another behavior optimized was the forward kick. The behavior was already in use prior to being optimized but we wanted to maximize the distance achieved by the ball. The fitness function is represented in equation 2. Again, the function $bell(x)$ is used to ensure that the ball is kicked strictly forward.



(a) Hill Climbing. (b) Tabu Search.

Fig. 3: Graphs of the fitness over the iteration reported in the optimization of the side kick skill for both Hill Climbing and Tabu Search algorithms.

$$f(x) = \frac{bdist_x^3 * bell(bdist_y)}{\Delta t} \tag{2}$$

Table 3: 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)

The table 3 shows the parameters used in the experience. The results of this experiment are reported in the table 4. The best results were achieved with Hill Climbing resulting in a kick that moves the ball 6.30 m. The skill duration was reduced from 2.04 s to 1.78 s. The graphs of figure 5 show the convergence of the optimization methods. The blue lines represent the fitness of the current experiment and the red lines are the best fitness until that iteration.

5 Conclusion

Humanoid robots are currently not able to perform as well as humans do some simple tasks, such as kicking a ball. This work offers an automated process of reducing this difference by improving existing behaviors with automatic optimization methods. The initial work was made to create an optimization framework

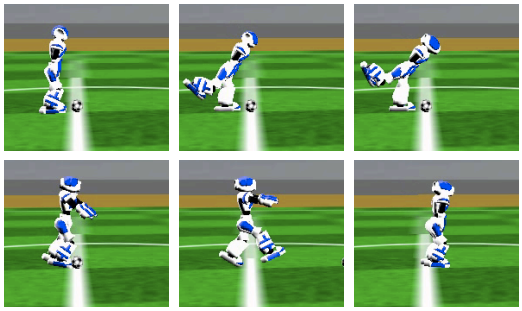
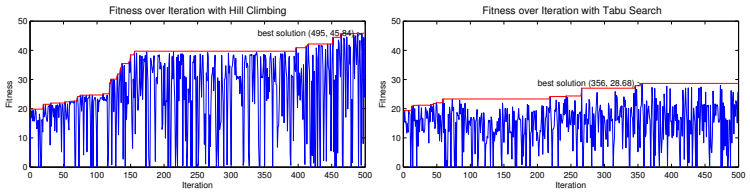


Fig. 4: Sequence of images showing the execution of the front kick skill (not optimized).

Table 4: 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



(a) Hill Climbing. (b) Tabu Search.

Fig. 5: Graphs of the fitness over the iteration for the longest distance kick optimization with the Hill Climbing and Tabu Search methods.

for the behaviors of the FCPortugal Team 3D humanoid agent which inhabits the simulated world of the RoboCup 3D simulation server. This required both modifications to the agent, the simulation server and the creation of the optimizer itself. The configuration of the optimizer was designed to be easily used with different kinds of behaviors. Also, the use of scripts to implement the objective function, makes the optimization much more generic and easy to configure. Two behaviors were optimized in this work: a side kick and a front kick. Both were created using the SlotBehavior model and the results were very good.

The side kick reached the best results with the TS algorithm, having an improvement of 91% with the ball distance being raised from 1.38m to 2.64m. The forward kick optimization also achieved good results: from 5.15 meters of ball distance now the kick can get 6.30 meters, which is an improvement of 22%.

References

1. Behnke, S.: Online trajectory generation for omnidirectional biped walking. In: Robotics and Automation, 2006. ICRA 2006. pp. 1597–1603. IEEE (2006)
2. Boedecker, J., Asada, M.: Simspark–concepts and application in the robocup 3d soccer simulation league. In: SIMPAR-2008, Venice, Italy (2008)
3. Campos, J.: Real-time well drilling monitoring using gocad. In: 22nd Gocad Meeting. vol. 10. Nancy (2002)
4. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: The nao humanoid: a combination of performance and affordability. Arxiv preprint (2008)
5. Ierusalimschy, R., De Figueiredo, L., Filho, W.: Lua-an extensible extension language. Software Practice and Experience 26(6), 635–652 (1996)
6. Ierusalimschy, R., de Figueiredo, L., Celes, W.: The evolution of lua. In: 3rd ACM SIGPLAN conf. on History of programming languages. pp. 2–1. ACM (2007)
7. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for ai. AI magazine 18(1), 73 (1997)
8. Lund, H., Miglino, O.: From simulated to real robots. In: IEEE International Conference on Evolutionary Computation, 1996. pp. 362–365. IEEE (1996)
9. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In: 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012) (June 2012)
10. Mouton, C.: A study of the existing libraries to read from configuration files (from c++). Arxiv preprint arXiv:1103.3021 (2011)
11. Picado, H., Gestal, M., Lau, N., Reis, L.P., Tomé, A.M.: Automatic generation of biped walk behavior using genetic algorithms. In: IWANN (1). LNCS, vol. 5517, pp. 805–812. Springer (2009)
12. Rei, L., Reis, L.P., Lau, N.: Optimizing a humanoid robot skill. 11th Int.l Conf. on Mobile Robots and Competitions pp. 84–89 (2011)
13. Shafii, N., Reis, L.P., Lau, N.: Biped walking using coronal and sagittal movements based on truncated fourier series. RoboCup 2010 pp. 324–335 (2011)