Automated Optimization of Walking Parameters for the NAO Humanoid Robot

Nicolò Girardi [a] Chiel Kooijman [a] Auke J. Wiggers [a]

[a] University of Amsterdam
Science Park 904
1098 XH Amsterdam

Abstract

This article describes a framework for optimising walking parameters for a NAO bipedal robot, both in a virtual setting and in the real world. It also covers preliminary results of the first experiments and provides suggestions for future research based on those experiments and the authors' experience from hand-tuning those parameters.

1 Introduction

The main focus of the project is improving the walking motion of NAO robots by optimizing a set of parameters for an existing walking engine. Our research is specifically about a game competition, the RoboCup Standard Platform League (SPL), but has also a broader value. Walking for (bipedal) robots is one of the main challenges in robotics. Improvements allows them to move faster and avoid damage to hardware or environment, and enables them to reach and function in places designed for humans. There is an even broader meaning in this work: To study the effectiveness of different machine learning methods in real life environments, which means taking into consideration factors such as field imperfections, sensor error, error in joint angles, and overheating of servos. To facilitate this research, a specific framework was created, which allows researchers to manage the learning process and obtain data for evaluation.

The RoboCup's aim is to promote Robotics and AI research in a playful and entertaining fashion that would appeal to non-expert people and potential future generations of researchers [5]. The event is also an occasion for insiders to share and expand their knowledge. The competition factor plays a role by creating an environment which stimulates improvement. The fact that contending in the SPL consists mainly of writing software, helps this sharing process, as code from the winning team has to be published after the end of the competition.

2 Related work

Multiple methods that combine elements of learning in real life and learning in simulation have been proposed. Haasdijk remarks that in online learning, the average performance of the population is more important than the best performing individual, as there is always a population, so each individual should perform well [3]. Bongard et al. suggest combining simulation and real-life evalualuation to find an approximate evaluation in the simulator before assessment in the real world. This way the poorest performers may be prevented from influencing the average performance [1]. Nordin and Banzhaf note that genetic algorithms can be used in an on-line setting, where the evaluation will be different for each individual, as the averaging tendencies even out the effects of probabilistic sampling, resulting in a set of good solutions [8].

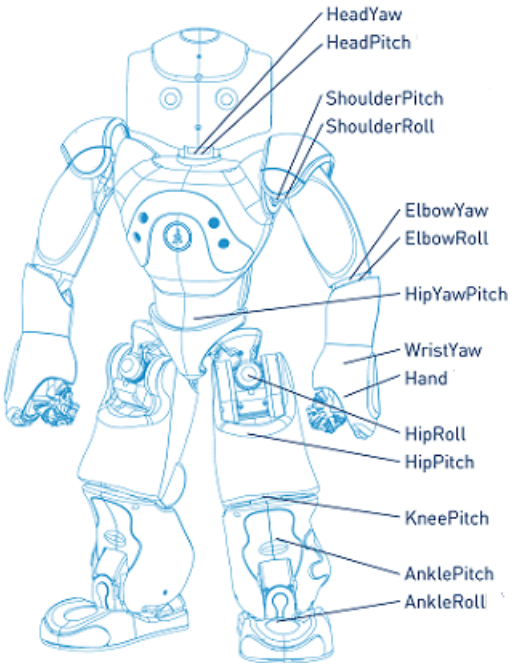

Figure 1: Aldebaran overview of NAO joints.

MacAlpine and Stone have been successful in the use of a genetic algorithm to teach a NAO robot to walk. Their approach uses the CMA-ES policy search, and did an extensive search of a policy space consisting of 14 out of over 40 parameters [6]. However, for this method to show any result, a huge number of rollouts (simulation runs) is needed. This is not necessarily true for some other gradient methods and genetic algorithms, which can show improvement early on (even before convergence). Both have been applied by Meijer, who tested performance of methods he refers to as Finite Difference, Vanilla, Episodic Natural Actor Critic and a genetic algorithm, in learning an optimal kicking motion for a NAO [7]. It should be noted that these methods were applied to improve performance of an already existing kicking motion.

2.1 Genetic algorithms

Genetic algorithms are a name for machine learning methods that aim to optimize through means similar to biological evolution. A basic genetic algorithm contains at least the following:

- Properties that define an organism: The ‘genes’ that can be altered to create new organisms.
- A population of organisms that is to be evaluated.
- A fitness function and corresponding evaluation tasks that specifies how these organisms are evaluated.
- A way to generate new generations based on old ones.
- Learning parameters that influence how new generations are formed.

The algorithm evaluates the population, and based on each organisms performance, selects ‘parents’ out of which new generations can be created, similar to the biological process of natural selection. Through the fitness function and corresponding evaluations, the search can be guided. A disadvantage of this method is that the fitness function must be user-specified, which may be hard to do in tasks with multiple objectives or in cases where the criteria for performance are unclear.

The ways in which new generations are generated are similar to biological generation of offspring:

- Transmission of parts of either parent.
- Crossover between two parents by taking a (weighted) mean of parts of the two.
- Mutation by adding random elements to the organism.

For each generation there can also be a part of individuals that is directly copied to the next one (the organisms survive). It's important to set a proper number of survivors and parents in order to have the highest quality features of fittest individuals pass to the next generation without however limiting the search space to a small area. If too many organisms survive, it could be possible that these and their offspring dominate the concurrent generations in a few timesteps, causing a local optimum to be reached.

3 Approach

3.1 Platform

The robot used in SPL is NAO (version H25) by Aldebaran [2], a humanoid, bipedal robot. It has inertia sensors that enable fall-detection, and a total of eight resistance sensors in the two feet that allow for early detection of imbalance. Both of these are used in the evaluation of parameter settings. Figure 1 shows an overview of the NAO's joints.

3.2 Walking engine

In the RoboCup 2013 we used the NaoTH framework for NAO by Berlin United [4] for various reasons: it features a modular code which makes it easy to expand, and provides a walking engine from which we can select a set of parameters for learning.

The walking system is closed loop, and uses a set of more than thirty parameters. A subset of this set was selected manually for learning, as these were likely to contribute to a stable and fast walk than others. Some examples are:

singleSupportTime Defines how long the NAO stands on a foot for during the walking. This determines the walking frequency.

stepHeight Defines the maximum distance between the ground and the base of the feet. If not too high improves stability, but given the low weight of the robots, if set over a certain limit, makes the robot bounce and actually lose stability. It also requires more work by the joints which translates in overheating and joints' efficiency decreasing.

maxStep$_{x,y,\theta}$ Defines the maximum length (for walking forward), width (for strafing left and right) and angle (for turning left and right). Given a certain singleSupporTime, those parameters determine the speed of the movements. If set too high the result is a loss in stability.

bodyOffset$_x$ The body's offset in x-direction (forward). Setting this to a positive number will result in positioning the body forward during walking, which may contribute to faster walking in the x-direction.

bodyPitchOffset Indicates how much the body leans forward during walking with respect to the hips and legs. Similar to bodyOffset in that it can contribute to fast walks in x-direction.

CoMHeight Height of the center of mass of the robot during walking. The lower the body will be, the more stable the walk. However, the load on the leg joints will be higher and steps must be much smaller than for high center of mass.

This problem is to be solved using machine learning methods, as setting and testing these parameters manually is a time-consuming task and does not guarantee selection of an optimal parameter set. The differences between robots (for example actuator and sensor error) cause parameter settings to have a different effect on each robot. Also, the use of machine learning (and, more specifically, reinforcement learning) to teach robots to perform complex tasks is a relevant topic in AI research, and is a problem that is not limited to NAO robots, but applies to many other situations.

4 Implementation

For the use of machine learning methods described in the previous section, the NaoTH framework was extended with a module 'MachineLearning', which serves as a generic input for different machine learning methods. It consists of multiple classes:

MachineLearning The module class itself, handles debug requests from users.

LearnToWalk The evaluation program, used by the module to test performance of specific parameter sets.

MachineLearningMethod An abstract class that contains pure virtual functions, to be implemented by classes corresponding to specific machine learning methods.

This module can be controlled through debug requests. Alongside the module, an interface was created for easy parameter selection and the display of information during learning.

4.1 Learning Parameters

To set the genetic algorithm we use specific parameters:

crossoverRate Defines the weight for crossover.

mutationRate Defines the weight for mutation.

trasmitRate Defines the weight for transmission.

maxGeneration The maximum number of generations that can be evaluated before the learning process ends.

ResettingTime Time before the Nao touches the ground after a previous evaluation has ended.

RunningTime Duration of an evaluation.

StandingTime Time during the Nao stays still after touching the ground before starting a new evaluation. It's used to guarantee stability for the body when starting an evaluation.

Number of survivors The number of individuals that are copied from a generation to the next one.

Number of parents The number of individuals from a generation that are used to obtain a new generation.

Size of population The number of individuals for each generation.

4.2 Evaluation Tasks

The evaluation tasks that were defined to run the tests are:

- Walk forward/backward
- Strafe left/right
- Turn right/left x degrees
- Walk diagonally (walk forward/backward + strafe left/right)

These tasks can be combined to obtain more realistic and challenging action sequences.

4.3 Fitness Function

The walking distance is at the base of the fitness function used in the experiments. To emphasise that the robot should be able to walk in any direction, each evaluation consists of a set of tests, which can include walking forward, backward, sideways (strafing), rotating, and walking diagonally in any direction, we use the weighted harmonic mean (equation 1). It rewards increases closer to 0 more than at higher values, which should result in a behaviour that should at least be able to perform each task to a certain minimum level.

$$\frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{x_i}} \tag{1}$$

The weights allow for focusing more on specific tasks such as walking forwards. The framework allows for giving a negative reward when the robot falls over. The individual can be killed to save time and reduce the chance of damage in the case of testing in the real world.

4.4 Graphical User Interface

The learning process is managed and controlled through a GUI implemented in Java (figure 2). Its main functions are:

Learn Starts the genetic algorithm

Select the learning method

List Shows the list of learning parameters currently in use

GetInfo Shows relevant information such as the current generation number, the current individual number, and the walking parameters for the fittest individual in the current generation.

4.5 Simulator

The used simulator is SimSpark [1], which is the official simulator for the RoboCup simulation leagues. It was designed to be able to simulate a large number of robots, which proves useful in machine learning tasks. A disadvantage of this simulator is that the models are different from the real robots, so that learned parameters can not be used in real life without some tweaking. A second disadvantage is that the simulator does not garantuee identical results for repeated simulation runs, as it is influenced by network delays on the machine that ruidenticalns the server.

[1] `http://simspark.sourceforge.net/wiki/index.php/Main_Page`

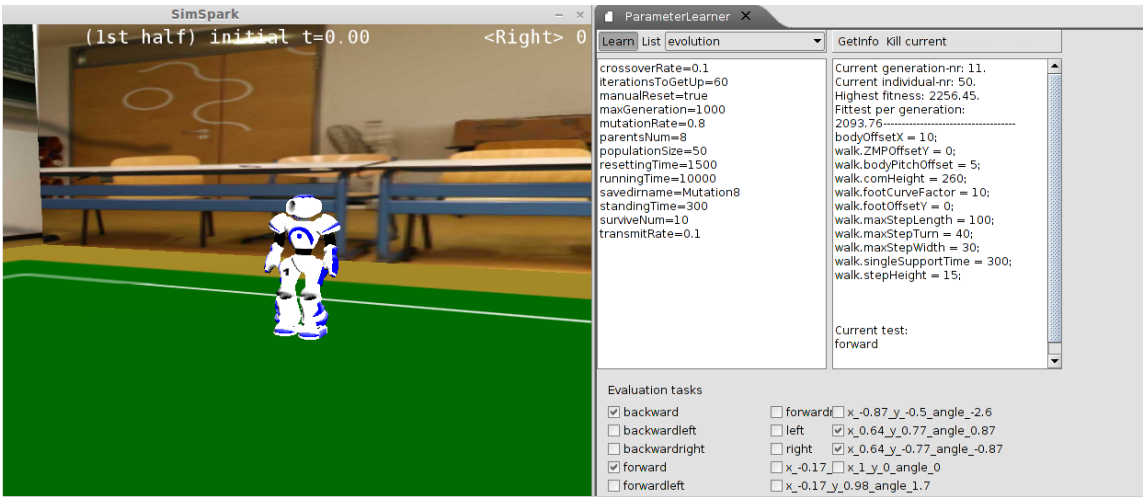

Figure 2: Example of the graphical user interface (right) and SimSpark (left).

5 Results

To test the performance of the implemented framework, two separate experiments were conducted. Both will be described below. For both of these, the performance on each task is measured by taking the traversed distance as an evaluation score. Falling down is penalized, and results in subtracting 1000 from fitness (empirically chosen), so that organisms that fall early in the episode are punished more severely than those that fall at the end of their task.

5.1 Experiment: Walking forward

To test the effectiveness of the method, it was first applied to improve the walk in the x-direction. The effect of different parameter settings can be seen in figure 3. The settings for these tests were as follows:

Figure	Transmission	Crossover	Mutation	# Parents	# Survivors	Populationsize
Mutation	0.1	0.1	0.8	10	10	50
NoMutation	0.5	0.5	0.0	10	10	50
Crossover	0.1	0.8	0.1	10	10	50
Equal	0.1	0.1	0.1	10	10	50

It should be noted that the first generation of each test contains only copies of the first individual, making the performance for this generation prone to error.

5.2 Experiment: Holonomic walk

In this experiment, machine learning was applied to improve the omnidirectional walk of the Nao. Evaluation was performed by forcing Nao to execute multiple different subtasks, which were to be performed in quick succession - with no pause in between tasks. The tasks were chosen so that they would closely resemble actions during a football match. There was no bias toward any of the evaluations, with each phase lasting 7 seconds, although the order of the tasks would always be the same. Falling down was penalized, and caused the episode to finish immediately.

A parameter set was evaluated based on performance in each of these tasks:

- Walking forward/backward
- Turning
- Walking towards a static target
- Quick stopping/starting and direction changes

The graph in figure 4 shows how the average fitness of the population changes over time. The graph suggests that the fitness is generally increasing in the first 25 generations. The graph in figure 5 shows the fitness of the fittest individual in the population over time, which is monotonically increasing. Reason for this is that the surviving group of each population, of which the fittest individual is part of, is not re-evaluated.

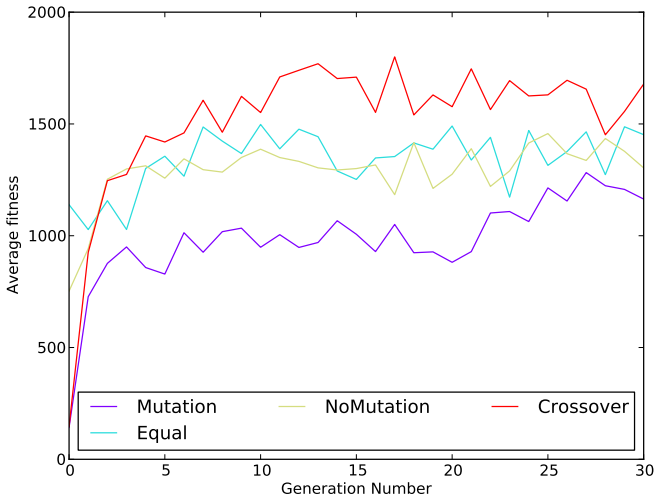

Figure 3: Average fitness for different parameter settings.

5.3 Evaluation

Results for both experiments indicate an incline in performance. As can be seen in figure 3, it proved useful to limit mutation, as random trials usually ended up in organisms with lower fitness than generations created using different settings. Test NoMutation is used as a benchmark, needed because of noise in the evaluation, by setting the mutationrate to zero. This ensures that each generation contained only exact copies of the very first organism. Its low performance at generation 0 can be explained by this noise as well, as the first generation contained only copies of the initial organism. Although setting transmission-, crossover- and mutationrate equal resulted in nearly the same performance, setting crossover-rate relatively high resulted in higher fitness.

The fitness function makes it so that performance in the second experiment can not be directly compared to the first experiment, however, the respective relative increases in performance can be compared. The second experiment shows a more steady increase in performance. This can be explained by the complexity of the task: The chance that a mutation improves performance in just the task of walking forward is higher than it improving performance in multiple subtasks, all of which contribute to the total fitness of the organism.

Results indicate that the selected method and chosen parameter settings improve mean performance of the generation over time and will continue to do so.

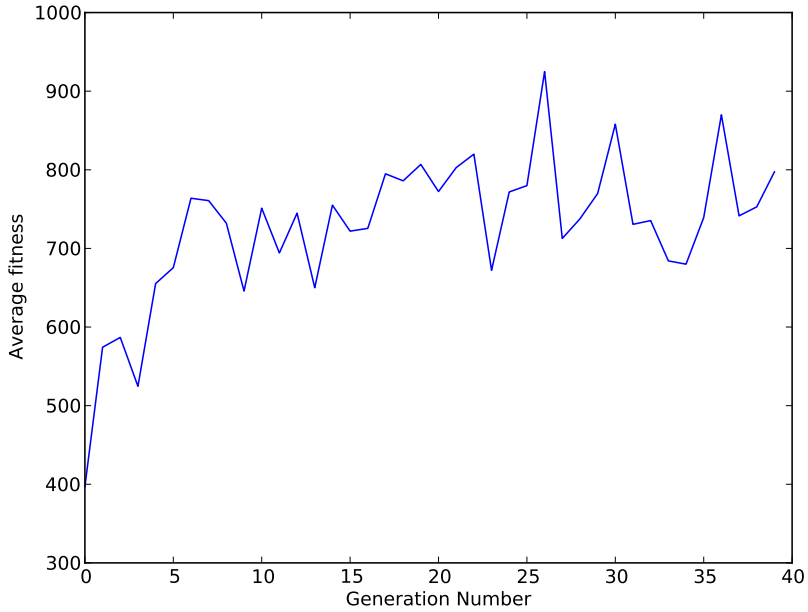

Figure 4: Average fitness of the population over time.

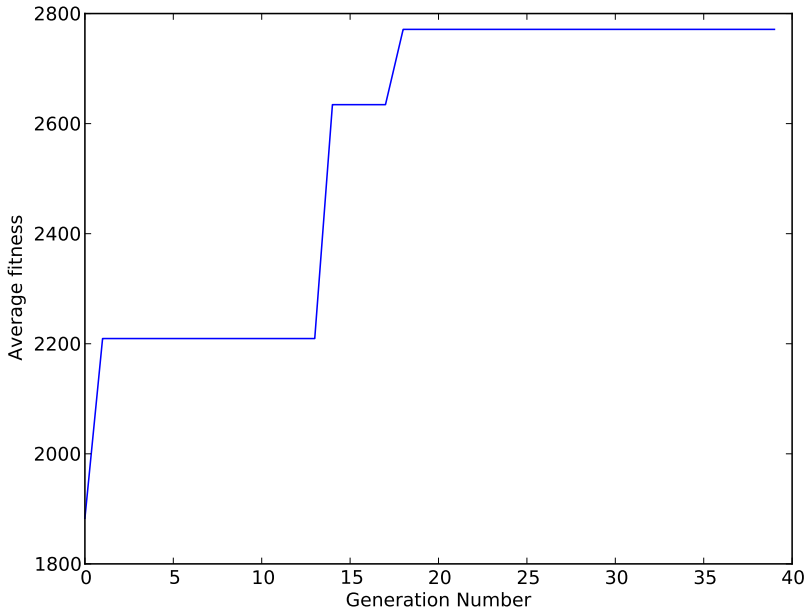

Figure 5: Maximum fitness of the population over time.

6 Conclusion

In this project, the effectiveness of evolutionary algorithms is tested by applying these to optimize walking parameters for a bipedal robot. At the start of the learning process, a working set of walking parameters is available in an existing framework, meaning that not the joint values but the parameters for the framework must be optimized. Different parameter settings are tried to learn the optimal parameter-set for a single task, namely walking forward. Based on this first experiment, the best found settings are used to learn an optimal parameter-set for a holonomic walk. Results for both experiments indicate that there is indeed an improvement in performance, but that starting with a suboptimal set makes it so that the resulting increase in performance is small.

7 Discussion and Future Work

Since old individuals are not re-evaluated and the evaluation function is noisy, it is likely that the population's evolution is inhibited after an individual has gained a high fitness by chance. Re-evaluating should even out irregularities in the fitness and increase the overall performance [8].

The framework is eventually to be extended to facilitate on online learning, specifically learning during matches. This would allow for more accurate evaluation of fitness, as the tests in the lab setting are not the same as a real match. It would also allow for more optimal usage of time during an event, as the robots must be allowed to cool down before matches, and differences in carpet make evaluation before arriving at a RoboCup event inaccurate.

An other suggestion is to classify different kind of actions with different parameter sets, as parameters that work well for walking forward seem to be different from parameters that perform well on other tasks. This would, however require changes to the framework in order to transition smoothly and without falling.

References

[1] Josh Bongard, Victor Zykov, and Hod Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[2] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jérôme Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of nao humanoid. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 769–774. IEEE, 2009.

[3] Evert Willem Haasdijk. Never too old to learn: On-line evolution of controllers in swarm-and modular robotics. 2012.

[4] Verena Hafner, Hans-Dieter Burkhard, Heinrich Mellmann, Thomas Krause, Marcus Scheunemann, Claas-Norman Ritter, and Paul Schütte. Berlin united - naoth 2013. 2013.

[5] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for ai. *AI magazine*, 18(1):73, 1997.

[6] Patrick MacAlpine and Peter Stone. Using dynamic rewards to learn a fully holonomic bipedal walk.

[7] Christiaan Meijer. Getting a kick out of humanoid robotics. 2012.

[8] Peter Nordin and Wolfgang Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140, 1997.