Technical Report

IRI-TR-17-03

On the application of CMA-ES to biped walk

Alejandro Suárez
Carles Miralles
Salvador Medina

January 2017

Institut de Robòtica i Informàtica Industrial

Abstract

This project studies the applicability of *Evolution Strategies - Covariance Matrix Adaptation* to enable a biped robot to walk with no previous knowledge. While most work on the subject is based on *Genetic Algorithms*, we believe that CMA-ES shows indeed great potential. Our work focuses on the *DARwIn-OP* robot, although it can be easily extrapolated to other humanoids. The evaluation of the solutions' goodness is performed via simulation to keep the real robot away from potential harm.

Keywords: Evolutionary Algorithms, CMA-ES, Robotics, Biped Walking

1. Introduction

Controlling humanoid robots to make them walk is a rather complex and difficult task. Among all the difficulties, one of the most notable ones is that the movement itself is unstable in its nature. The robot is not fixed to the ground and in each walking cycle there are two moments in which the robot is standing in just one foot while the other is moving to perform a step. Several disciplines like Physics, Biology, Mechanics and Computer Science converge in the understanding of the sequence of movements that lead to a successful and effective gait. Our purpose is to apply CMA-ES to come up with good solutions for this challenge.

1.1. Robots and simulation frameworks

The evolution until achieving a successful gait behavior goes through many failed attempts (i.e. falls) so it is dangerous and unfeasible to work directly with robots. Therefore the use of simulators is a must. We make use of a very realistic simulation model of our humanoid robot. It goes without saying that this solution can be easily extrapolated to other robots, provided their model is available. We use ROS and the Gazebo simulators. More details on this can be found in Appendix A.

The selected robot is a *DARwIn-OP*: a small humanoid robot manufactured by ROBOTIS (the interested reader may see the specifications in Appendix C). This particular robot is an open platform and has been used in several competitions like *IEEE ICRA Robot Challenge*[1] and *RoboCup*[2]. In addition *IRI*[3] will provide a physical one subject to satisfactory results.

1.2. Goals of our work

The movement of a robot is determined by the evolution of the joint values that are associated to the robot's actuators. Therefore our main objective is to find a set of joint's functions for each of the *DARwIn-OP*'s actuators that allow it to walk. Moreover we would like to make the *DARwIn-OP* advance as fast as possible. Once a solution has been found, the next step would be to test it on the real robot.

We propose ourselves, as a milestone, to make the *DARwIn-OP* walk faster than in the specifications (i.e. 24cm/s as shown in Appendix C). Also, we would like this solution to be very stable and reproducible.

1. ICRA 2012 Home Page: http://icra2012.org/program/robotChallenge.php
2. RoboCup Mexico 2012: http://www.robocup2012.org/
3. Institut de Robtica i Informtica Industrial: http://www.iri.upc.edu/

We believe that CMA-ES can yield faster results than the more commonly used GA in this application. We pose as a secondary objective to find a configuration of the algorithm (fitness function, the initial strategy's centroid and the CMA-ES's parameters) that leads to a fast convergence. We believe that a faster convergence would make Evolutionary Algorithms' approaches to this challenge more attractive and widely used. It would allow for a quicker adaptation of the gait behavior of the robot under small variations such as changes in the mass distribution, in the PID controllers that govern the actuators or in the physic properties of the environment. Nevertheless checking the performance in these special environments is outside the scope of this project.

2. Previous work

The robots' presence in our daily lives is nowadays more noticeable and is estimated that their use will explode in the current century. Humanoid robots present a great potential when it comes to adaptability. Therefore finding methods that allow to find the parameters needed for optimizing (under different criteria) biped locomotion is still an interesting research topic with important practical implications. The current section reviews some of the different approaches tried in the past.

The work in [Pratt, 1995] describes a technique called Virtual Model Control. It consists in a control scheme based on a high accuracy robot model and the interaction and constraints between different parts of the robot. This method is quite robust since it relies on a precise model of the robot and on its physical characteristics to calculate the movement parameters. Therefore it is a robot dependent method. Another difficulty addressed in the dissertation is the non-ideal actuators (noise, latency, etc).

Another method is that of [Benbrahim and Franklin, 1997] which uses Reinforcement Learning, namely the self scaling reinforcement algorithm (SSR). In this case the robot learns to walk by means of repeated experiments in which the actions that lead to failure (e.g. falling or not meeting another soundness criteria) are penalized, while the actions that lead to successful behavior are rewarded. This approach gains in generality and adaptability with respect to the previous one at the price of introducing a learning stage before the robot can actually walk. However, according to [Benbrahim and Franklin, 1997, p. 64], the use of reinforcement learning without any previous knowledge takes impractical times. To overcome this issue, a supervised learning approach is proposed by means of adding a pre-trained neural network with examples obtained from a simulator environment. This adds one additional step of supervised learning to the solution.

We can find several other authors that apply generic methods. In the same line as us, applying Evolutionary Algorithms, we can find [Picado et al., 2009], [Arakawa and Fukuda, 1996] and [Heinen and Osório, 2006]. These three examples use Genetic Algorithms, but considering different soundness criteria (i.e. fitness function), solution representation (the Fourier oscillators from [Picado et al., 2009] vs the joint sequence specification of the other two papers) and scope. In particular, one of the most notable differences is that the last of these papers considers robots with more than 2 legs (which are actually more stable). In essence, the common motif behind these last three works is that the individuals (chromosomes) determine the movement of the robot's joint. Each of these individuals is simulated and the fitness function indicates its performance.

[Hebbel et al., 2006] falls into the category of Evolutionary Algorithms as well. In fact it is closer to our own approach since it considers both Genetic Algorithms and several flavors of Evolution Strategies (although not CMA) like $(1+1)$ with the 1/5 rule, $(1, 30)$, $(5, 30)$ and $(5+30)$. The authors propose using a parametrization of the curves described by the feet as the individual representation.

Perhaps surprisingly, there are not many works that revolve around the application of CMA-ES to gait-behavior generation. In a slightly related way [Farchy et al., 2013] uses CMA-ES to optimize the parameters of a simulator, so the behavior of the simulated robot is as faithful as possible to the behavior of the real one. Then, the simulator is employed to optimize the walking behavior of the robot and the cycle starts again. In this approach, simulation and testing on the real robot are more closely interleaved. Contrarily to us, the individuals of the CMA-ES algorithm are not robot's movements specifications, but simulation parameters that are adjusted to represent faithfully the behavior of the robot.

3. Application of CMA-ES to biped-locomotion generation

For this project, we have decided to use a particular subfamily of Evolution Strategies: Covariance Matrix Adaptation - Evolution Strategies (CMA-ES). The pillars that support this decision are mainly three: (1) at the time of writing this report, this algorithm is considered the state-of-the-art of ES (arguably even of Evolutionary Algorithms); (2) the lack of literature on the application of CMA-ES to tackle the generation and optimization of biped locomotion; (3) and our genuine interest in discovering how well-suited it is for this particular task.

CMA-ES was first described in [Hansen and Ostermeier, 2001]. In words of their authors, it does not require an intensive parameter tweaking. For the reader's convenience, we have outlined the CMA-ES algorithm (pseudo-code included) in Appendix E. The new individuals are generated according to a multivariate Gaussian distribution. Part of the success of this method comes from the great power that provides the Covariance Matrix Adaptation scheme to adapt to the fitness function. This adaptation is somewhat reminiscent of the update step in quasi-Newton methods, although derivative free, robust to noise and with potential to avoid getting stuck in local extrema.

The algorithm depends on the parameters λ, μ, as well as the initial m, σ_0 and C. Here λ is the size of the population each generation; μ defines the number of individuals that will be used to calculate the new mean; m is the centroid of the newly generated individuals; σ_0 represents is the "step" and modulates the strength of the mutation; and C is the initial co-variance matrix. Note that σ_0 and C are updated at each generation depending on the generated population and that their initial value can be decided (or let to the default).

3.1. Fitness function

The parents of the next generation are decided depending upon the fitness function. This function is very important, as it guides the algorithm to a particular kind of solutions.

We use the fitness function proposed in [Heinen and Osório, 2006]:

$$f = \frac{\delta}{1+\theta} \tag{1}$$

where

$$\theta(r_{1\ldots n}, p_{1\ldots n}, y_{1\ldots n}) = \sqrt{Var(r_{1\ldots n}) + Var(p_{1\ldots n}) + Var(y_{1\ldots n})} \tag{2a}$$

$$\delta(y_{leftfoot}, y_{rightfoot}) = \max(y_{leftfoot}, y_{rightfoot}) \tag{2b}$$

δ represents the distance traversed by the robot during a single simulation (we simulate 10 seconds). Instead of using the center of gravity or any other part of the body, we use the distance of the furthest foot in good contact with the ground. This will ensure that individuals that fall forwards are not preferred over individuals that barely advance but do not fall. This will also avoid no-biped advancement up to certain extend.

θ measures the instability of the robot. Namely, it is the average oscillation (row+pitch+yaw) of the torso in radians.

Combining δ and θ as shown in Equation 1 we can optimize both the robot's speed and stability avoiding multi-objective fitness function. This way we do not incur on a careful selection of weighting factors for each objective. Moreover for this first take on the problem we want to avoid including very specific knowledge about the robot and the walking dynamics (like the ZMP in [Arakawa and Fukuda, 1996]).

It is important to stress that the fitness function is not deterministic because the simulator has a stochastic behavior[4]. Therefore the same individual will render different fitness values each time it is simulated. Typically, the deviation is quite slight, but there are cases in which the individuals are at the edge of stability and may either fall or walk at a fast pace.

Therefore, in addition to the fitness function from equation 1, we propose an alternative fitness evaluation that emphasizes stability and is robust against randomness. This evaluation can be seen in Algorithm 1. Basically it consists of: (1) performing several simulations; (2) picking the worst of the measured fitness (worst-case scenario); (3) and clamping the distance so once the robot reaches δ_{max}, the only way of increasing the fitness function is by reducing the average oscillation.

input : individual I
output: Fitness evaluation
1 **for** $i \leftarrow 1, M$ **do**
2 $\quad \delta_i, \theta_i \leftarrow$ `SimulateIndividual(`I`)` ;
3 $\quad f_i \leftarrow \frac{\texttt{min}(\delta_{max}, \delta_i)}{1+\theta_i}$;
4 **end**
5 **return** `min(`$f_1 \ldots f_M$`)`

Algorithm 1: Alternative fitness evaluation procedure

We apply the fitness function from Equation 1 in section 4.1 to generate walking behavior starting from static positions. Then, as we can see in section 4.2, we use Algorithm 1 as the evaluation method in an improvement phase aimed at generating more stable individuals starting from marginally stable ones.

4. Simulations in Gazebo are not repeatable: http://answers.ros.org/question/40208/gazebo-simulations-not-repeatable/

3.2. Representation of the solutions

We apply the idea proposed in [Picado et al., 2009]. Since joints typically follow a cyclic movement, we can associate a periodic oscillator to each of them. Each oscillator consists of a single term Fourier series (i.e. a constant plus a sinus function), as shown in 3. Much like in the aforementioned work we assume that all the oscillators share the same period.

$$f(t) = C + A_1 \sin\left(\frac{2\pi}{T}t + \phi_1\right) \forall t \in R \tag{3}$$

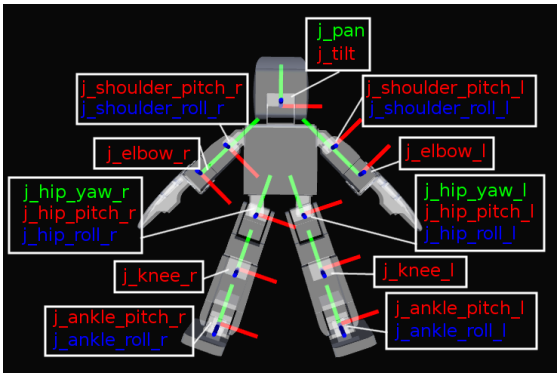


Figure 1: This image depicts all the *DARwIn-OP*'s joints.

This means that for defining a solution using this simplification we have to consider 3 variables for each joint: the offset C, the amplitude A and the phase ϕ, as well as the period of the global oscillation T.

In the case of the *DARwIn-OP* robot, we would need $N_{joints} \cdot 3 + 1 = 61$ parameters to represent the movement of all the joints. We can see in figure 1 all the robot's joints. Fortunately, the number of parameters can be greatly reduced if we fix the joints that do not have a great influence over the gait (like *j_elbow*) and we assume sagittal symmetry.

For the purpose of this work, each individual explicitly defines the movement only for *j_shoulder_pitch_l*, *j_hip_roll_l*, *j_hip_pitch_l*, *j_knee_l*, *j_ankle_pitch_l* and *j_ankle_roll_l*. The right hand side is inferred by symmetry. For this reason, the total number of variables needed for representing a solution is $6 \cdot 3 + 1 = 19$ real values.

It is worth mentioning that we define a maximum and minimum limits for each of these 19 parameters to keep the algorithm from wasting time on absurd solutions. If a generated individual violates some of these constraints, we forced it back (reproject) into the feasible space. This is one of the methods studied in [Diouane et al., 2015] for dealing with constrained problems.

3.3. Termination criteria

Problems with a big search space and noisy fitness functions like this one may take a lot of time to converge, or even not converge at all. Furthermore, one of the main goals of our work is to be able to find a solution in a very small time. For this reason, it is very important to accurately define the termination criteria of the executions.

We have picked four of the criteria described in [Hansen, 2009]. The complete list of termination criteria that we have used is listed below:

- **MaxIter.** We have modified the formula from the paper. Considering that a normal laptop computer can execute up to 4 simulations in real time, that each simulation

takes up to 10 seconds and we want to limit this time to 3 days, the maximum number of individuals that can be generated is about 100000. For this reason, we will limit the maximum number of iterations to $100000/\lambda$.

- **Stagnation.** The median of the 20 newest values is not smaller than the median of the 20 oldest values, respectively, in the two arrays containing the best function values and the median function values of the last $0.2t + 120 + \frac{30D}{\lambda}$ iterations. This criterion never triggered in our case.
- **ConditionCov.** The condition number of covariance matrix exceeds 10^{14} (inversion becomes numerically unstable).
- **EqualFunVals.** In more than $\frac{1}{3}$ of the last D iterations the objective function value of the best and the k-th best solution are identical, that is $f(x_{1...\lambda}) = f(x_{k...\lambda})$, where $k = 1 + 0.1 + \frac{\lambda}{4}$ (low diversity).

In addition, we have added an additional criterion of our own in the improvement phase described in 4.2:

- **GoodEnough**: the fitness value of the best individual from last generation exceeds $\delta_{max}/1.2$ (i.e. it reaches the threshold distance with an average oscillation below 0.2 radians).

4. Results and Discussion

In this section we describe the results we obtained from the experiments[5]. These results are compared with the ones obtained by [Picado et al., 2009] using Genetic Algorithms, since the characteristics of the robot used in his work are very similar to *DARwIn-OP*'s (similar size, mass and the same number of joints).

4.1. Starting from a statically stable individual

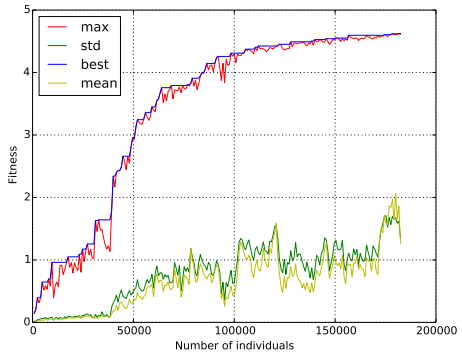
The first experiment that we have carried out is to check whether or not it is possible to learn good walking parameters starting from a static individual.

In order to do this, we have executed CMA-ES with the fitness function from equation 1, and the first four termination criteria specified in 3.3. We have tested different values of λ, μ and σ_0. Table 1 in annex F summarize our results. We consider solutions with fitness greater than 2 as good solutions since the average speed in these solutions is similar or greater than that of *DARwIn-OP*'s specification.

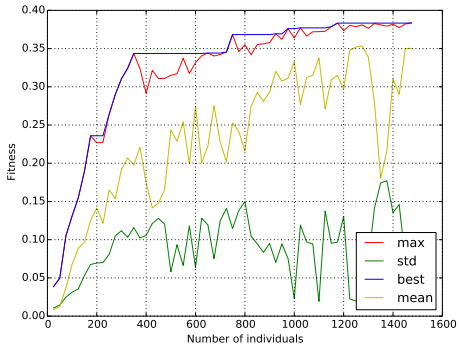
As it can be clearly seen in the tables, the algorithm usually finds better solutions for large values of λ. This means that a big number of descendant must be generated in each iteration to ensure that the algorithm does not gets stuck in a local maxima.

Another interesting noticeable fact is the influence of the μ parameter. CMA-ES tends to be faster and find better solutions for very small values of μ. This means that for the

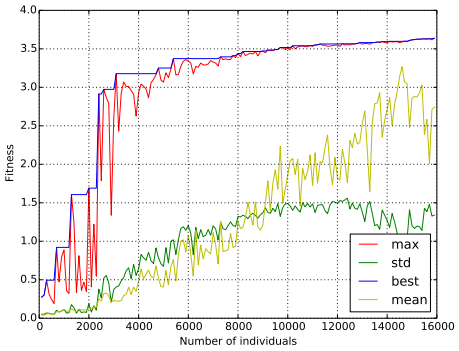
5. In addition to this section, the interested reader can check out a demonstration video of our work: https://www.youtube.com/watch?v=8n8mF-RmsNs



(*a*) Evolution of the fitness value for $\lambda = 800$, $\mu = 1$ and $\sigma = 0.10$ starting from an static pose



(*b*) Evolution of the fitness value for $\lambda = 25$, $\mu = 5$ and $\sigma = 0.10$ starting from scratch.



(*c*) Evolution of the fitness value for $\lambda = 100$, $\mu = 5$ and $\sigma = 0.15$ starting from an unstable solution.

Figure 2: Evolution of the CMA-ES starting from scratch and starting from an unstable solution. 2(*a*) (high λ) shows convergence to a quite good solution (*MaxIter* condition met), while 2(*b*) (low λ) shows very poor results (*EqualFunVals* condition kicked in). 2(*c*) corresponds to an improvement execution over the solution found in 2(*a*)

individual representation that we have defined, combining several good individuals does not help generating better descendants.

An example of a successful evolution can be appreciated in figure 2(*a*). One of the solutions that have proven to be better both in terms of the obtained fitness and visual verification is the one obtained with $\lambda = 800$, $\mu = 5$ and $\sigma = 0.10$.

We see that both the maximum and the average fitness improves slowly, until the maximum fitness reaches a value close to 1. From that point it increases fast until it approaches 4. At this point the curve flattens. Looking at the simulator, we can observe that in the first generations of individuals the robots always fall after a few steps, or fail to advance at all. This leads to a fitness that is similar among all individuals (low variance). After

the generation of some individuals that are able to walk for a small period without falling, the algorithm focuses on improving that solution, and the diversity and variance of the population increase. After that point, the best solution is further improved until the fitness value flattens.

This behavior also manifests with other parameters and executions, but the local maxima that the fitness function reaches varies. This is to be expected as CMA-ES is a stochastic algorithm and different executions might yield different results even with the same parameters.

On the other hand, figure 2(*b*) exemplifies a evolution process that gets stuck in a poor region of the solution space. This execution has been obtained with a comparatively low λ parameter (25).

4.2. Refining a marginally stable individual

In the previous subsection, we discussed the feasibility of making the robot walk starting from a static pose and without previous knowledge. In this case, it takes quite a long time to find the first promising solution.

A possible option to accelerate this process is to initialize the algorithm with a previously found individual as the initial mean. In figure 2(*c*) we show the fitness curves when the best individual of the last generation of a previous execution is used as the initial mean.

The fitness function has been slightly modified with respect to the previous experiment restricting the maximum distance to 4 meters and evaluating each individual 3 times taking the minimum fitness. With this modification of the fitness function we obtain slower (close to 0.4m/s) but more stable solutions. This is, we refine the already found sub-optimal solution.

The evolution of the fitness value in this case is different from what we saw for the static case. As we can see in the figure, convergence is reached much faster, with just 16000 individuals. The mean fitness increases fast and ends up very close to the maximum fitness value while its standard deviation stabilizes or even decreases. At this point, we see that all individuals in the last generations are almost identical and the *EqualFunVals* stop condition is triggered.

It is also relevant to point out that even though the fitness of the sub-optimal solution used as the mean was high, the mean and maximum fitness obtained in the first generations is rather low. This is due to the fact that the initial variance used is high. Even small variations in the parameters of the oscillators can make the robot fall and greatly harm the fitness value.

4.3. Testing our solutions in a real *DARwIn-OP* robot

As the last step of our experiment, we selected some of the more promising solutions and compiled them for being used in an actual *DARwIn-OP*. Sadly, none of these solutions worked and in some cases the robot walked backwards or fell. These are the possible reasons for this unexpected behavior:

- The robot that we were given was missing some parts (hands and protective case) and some of them were not the same of the specification (the original head was replaced by a camera) hence altering the mass distribution of the robot.

- The parameters of the simulation were set so that the friction between the floor and the robot's feet was high. In the real test environment, the floor was quite slippery. We tried to solve this by attaching rubber to the base of the feet but the robot kept falling.
- The simulation did not take into account the acceleration phase, present and determinant in a real world scenario.
- The script that we executed in the robot was not exactly the same as in the simulator, as we had to re-code it to *c* language. With just one session with the robot we did not have time to debug our code.

Some images of the experiments performed with the robot as well as a link to the video can be found in Appendix 4.3.

5. Strengths and weaknesses

In section 2, we have seen approaches like [Pratt, 1995] that are capable of getting a precise solution at the price of making explicit use of robot. This make these solutions robot-dependent. In our solution, those models are not part of the algorithm but just the simulator framework. By proceeding this way, our solution can be easily applied to other kinds of robots by just altering the model used in the simulator and the joint's names.

More advanced techniques such as [Benbrahim and Franklin, 1997] use supervised learning to get an robot-agnostic solution. To make this method feasible, the author explores the use of a pre-trained neural networks to provide prior knowledge and hence accelerate the exploration. CMA-ES provides a clear advantage over the aforementioned method since no previous knowledge nor supervised learning is required.

When we compare the use of CMA-ES with other works that make use of Genetic Algorithms, such as [Picado et al., 2009] and [Arakawa and Fukuda, 1996], we also find other advantages: with CMA-ES, definition of selection, crossover and mutation criteria is not required. In addition to that, some of the algorithm's parameters are subject to self-adaptation making it much easier to be tuned.

Moreover in terms of convergence velocity, we have obtained results that suggest that CMA-ES has the potential to find acceptable solutions in less time. For instance, in [Picado et al., 2009] the experiments needed 300 generations of 100 chromosomes each to come up with an individual that walks at 50cm/s. As we can see in table 2, an execution of CMA-ES $\lambda = 100$, $\sigma = 0.1$ and $\mu = 5$ yields a fitness of 4.1 (speed greater than 41cm/s) at the last generation (33700 individuals simulated), while it needs just about 8900 individuals to come up with a solution that is able to walk more than 2 meters. While we have not improved their mark, we have not devoted as much effort to this as we have to ensure that the gait is stable. However, we strongly believe that this milestone can be achieved.

Another strong point of our solution is that we can easily adapt the trade-off between speed and stability with the alternative fitness evaluation proposed in algorithm 1. This refinement phase can be performed in very few iterations, as we could also prove. It can be also useful to adapt gait parameters that work for certain conditions (environment, mass distribution, PID controllers) to other conditions.

There are also multiple drawbacks of this approach being the non-deterministic nature of CMA-ES one of the most troublesome (although the same can be said of any other EA approach). Even with good parameters, the algorithm might find different solutions in successive executions. The value given by the fitness function is not deterministic neither, as it depends on the simulator.

Another limitation of our work is the fact that we do not consider the gait-preparation phase and just focus on constant speed movement. Furthermore, the Fourier expansion used to describe the periodic movement of the motors might be too simplistic.

Also, we have not explored other external factors like differences in ground friction or inclination, tolerance of the motors or even the effects of wind. It would be possible to execute the algorithm for each of these scenarios with just slight modifications though.

6. Conclusions and future work

We have proposed the use of CMA-ES to optimize biped locomotion. We believe that this algorithm is, from theoretical and practical point of view, very suitable for the stated challenge because we have to navigate through a space of continuous solutions. Another reason that supports this choice is that CMA-ES is robust against the inherent noise present in most simulators (which is a good preparation for what awaits us in the real world). We have found several individuals that yield promising results inside the simulation environment. However there is still much work to do to cover the gap between the simulation and the real robot.

The experimentation part has made us aware of the captivating nature of evolution-based algorithm. Many unexpected results have shown up while working on the implementation: individuals that advanced crawling or adopting a quadruped pose. We could avoid this up to certain extent being careful on the way we measure the distance traveled by the robot and selecting reasonably low σ_0 parameters ("step" parameter of the CMA-ES algorithm).

At the moment of writing these conclusions we have covered only the case of an already marching robot, with no preparation steps to start the gait process. We had to take this into account via in the initialization of the robot's position at the beginning of each simulation.

Regarding CMA-ES particularities we have checked the repercussion of σ_0, λ (number of descendants) and μ (number of members of each generation). At the very beginning we started using big values of σ_0 getting as a result a chaotic execution of the algorithm that usually lead to crawling robots (seal-like movement). On the other hand, and although more rigorous statistical study is needed (i.e. more executions repeating the same set of parameters and analyzing them in terms of means and standard deviations), we have found a general pattern: we need reasonably high λ values (say, greater than 100), and quite low μ values (say, lower or equal than $\lambda/16$) to have certain guarantee of obtaining good solutions. We hypothesize that we need to generate enough descendants so the evolution does not get stuck following falsely promising directions, and that recombining too many individuals that are good on their own does not lead to better individuals.

An additional idea for future work is to introduce more specialized knowledge in the fitness function, although part of the appeal of this approach is precisely avoiding this. It can also be interesting to test CMA-ES with different types of solution (e.g. considering more terms in the Fourier expansion or considering completely different functions), and

to include somehow the optimization of gait-preparation procedure so the robot can walk starting from a completely static pose. Moreover we could face more difficult challenges like climbing stairs using CMA-ES.

References

Takemasa Arakawa and Toshio Fukuda. Natural motion trajectory generation of biped locomotion robot using genetic algorithm through energy optimization. In *Systems, Man, and Cybernetics, 1996., IEEE International Conference on*, volume 2, pages 1495–1500. IEEE, 1996.

Hamid Benbrahim and Judy A Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3-4):283–302, 1997.

Y Diouane, S Gratton, and LN Vicente. Globally convergent evolution strategies. *Mathematical Programming*, 152(1-2):467–490, 2015.

Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 2013 international conference on autonomous agents and multi-agent systems*, pages 39–46. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

Nikolaus Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2389–2396. ACM, 2009.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Matthias Hebbel, Ralf Kosse, and Walter Nistico. Modeling and learning walking gaits of biped robots. In *Proceedings of the Workshop on Humanoid Soccer Robots of the IEEE-RAS International Conference on Humanoid Robots*, pages 40–48, 2006.

Milton Roberto Heinen and Fernando Santos Osório. Applying genetic algorithms to control gait of physically based simulated robots. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1823–1830. IEEE, 2006.

Hugo Picado, Marcos Gestal, Nuno Lau, Luis P Reis, and Ana M Tomé. Automatic generation of biped walk behavior using genetic algorithms. In *International Work-Conference on Artificial Neural Networks*, pages 805–812. Springer, 2009.

Jerry E Pratt. *Virtual model control of a biped walking robot.* PhD thesis, Massachusetts Institute of Technology, 1995.

Appendix A. Implementation details

We have chosen *Gazebo*[6] as our simulation environment because it is easily integrable into ROS[7]. It has many adepts and supports the widely used URDF (Universal Robot Description Format) in which we can find the specification of many robots. We do not start from scratch: we make use of *DARwIn-OP*'s model developed by the IRI. We have adapted this model to our own needs (e.g. disabling the camera and the other sensors since we do not use as they add significant overhead to the simulation). In figure 3 we can see two instances of the simulator running in parallel.

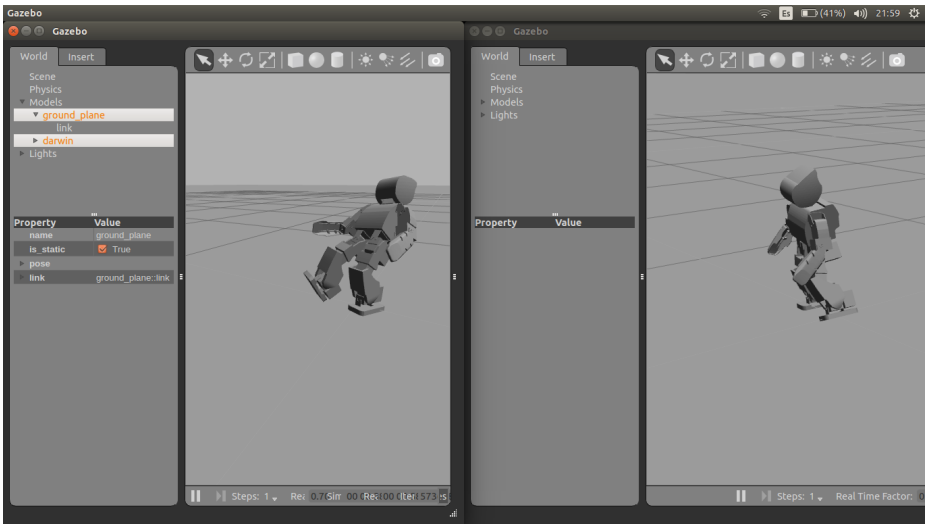


Figure 3: Two simulations running at once

We have used the DEAP's[8] utilities and implementation of CMA-ES to perform our work.

All in all, our implementation contains both *C++* and *Python* code.

A.1. Source

The complete source code that conforms our implementation, along with setup instructions can be found at https://gitlab.com/cmirallesp/ci.

Next we list all the scripts and modules that we have developed for this project. This list only contains the files that have been used for the final experiments and does not include previous tests. These test files can be found in the project's repository.

- **darwin_fourier_controller.cpp** A *Gazebo-ROS* plugin that gets the list of parameters of the oscillators and takes care of the joint trajectory execution (using an error-proportional or P controller). In addition to that, it computes the distance traversed

6. http://gazebosim.org/
7. *Robot Operating System* (http://www.ros.org/), an open source platform for working in the field of robotics with many utilities.
8. *Distributed Evolutionary Algorithms in Python*, http://deap.readthedocs.io/en/master/index.html

by the robot as well as the oscillation of the center of masses according to our fitness function definition in 3.1. In addition to this, the plugin detects the moment when the robot falls so that the simulation can be stopped before the specified maximum duration.

- **es_darwin.py** A python script that executes the CMA-ES algorithm with the specified values of λ, σ_0, μ and initial centroid. The number of *Gazebo* simulators running on the machine can also be specified with the *n_workers* parameter (we take advantage of the independence between fitness evaluations to run several simulations at once). This script generates checkpoint files at each generation (checkpoint_*_*_*.pkl and population_gen_*.pkl) so that if the execution is halted, it can resumed from the point where it was interrupted. It also implements the termination criteria specified in 3.3.

- **eval_pool.py** This script is used by *es_darwin.py* to communicate with the simulators. It defines the **FitnessEvaluationPool** class, which distributes the evaluation task among all the available *Gazebo* simulators.

- **bounds.py** Defines the limit of the parameters of the different oscillators according either to the specifications of the real *DARwIn-OP* robot or to what seems reasonable for a successful gait. This method is used when a new generation of individuals is generated by *es_darwin.py*.

- **sweep.py** This script iteratively executes the *es_darwin.py* script for different values of lambda, mu and sigma.

- **extract_population_data.py** This script reads the checkpoint files generated by *es_darwin.py* and plots the evolution of the fitness function. In particular, it plots the maximum fitness for all previous generations, the maximum fitness for the current generation and the mean fitness of the generation with its standard deviation.

- **reproduce_population.py** For a given checkpoint folder, this script reproduces the k best individuals of the specified generation in the *Gazebo* simulator.

- **get_table_values.py** Reads the checkpoint files of the specified folder and prints a table in LaTeXformat with the maximum fitness of the last generation, the number of individuals generated and the numbers of individuals needed to get a fitness value greater than 2.0.

Appendix B. Execution environment

For the simulations, we have used three machines with the exact same technical specifications. The most relevant ones are listed below:

- **Processor:** Intel i5-4460

- **Memory:** 4 x 4GB DDR @ 1866MHz

- **Storage:** Samsung SSD 850 EVO 250GB

- **OS:** Arch Linux 64bit
- **Environment:** Ubuntu 14.04 Docker Image

The simulation environment was packed in a Docker[9] image and distributed among the different machines. This made sure that the configuration and versions in all machines were exactly the same and greatly reduced the deployment time.

Appendix C. *DARwIn-OP*'s specification sheet

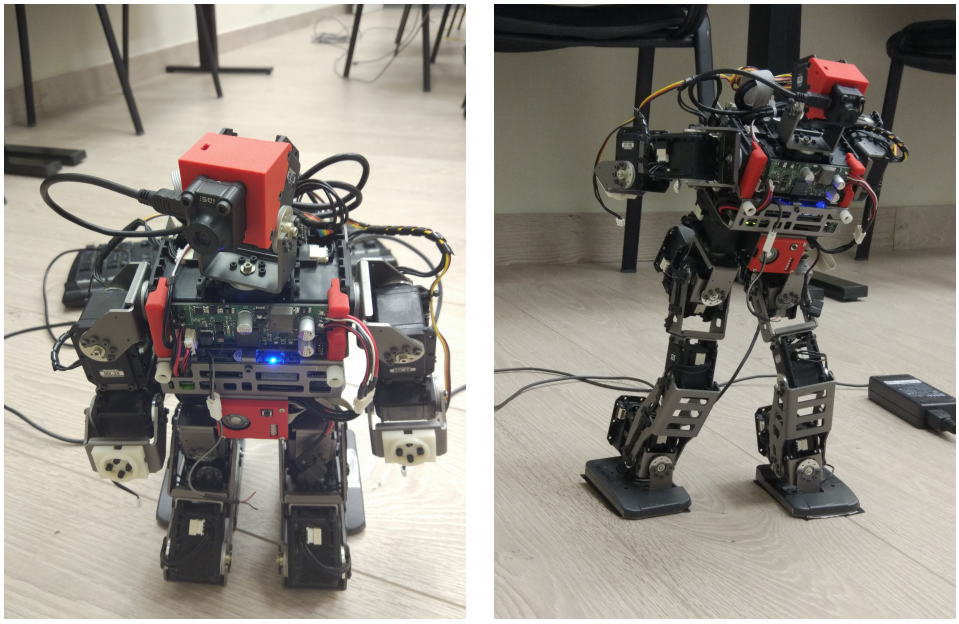
The following list details the technical specifications of the robot used for this practice. These specifications are extracted from its official website.

- Default walking speed: 24.0 cm/sec (9.44 in/sec) 0.25 sec/step - user modifiable gait
- Default standing up time from ground: 2.8 sec (from facing down) and 3.9 sec (from facing up) - user modifiable speed
- Built-in PC: 1.6 GHz Intel Atom Z530 on-board 4GB flash SSD
- Management controller (CM-730): ARM CortexM3 STM32F103RE 72MHz
- 20 actuator modules (6 DOF leg x2+ 3 DOF arm x2 + 2 DOF neck)
- Actuators with durable metallic gears (DYNAMIXEL MX-28)
- Self-maintenance kit (easy to follow steps and instructions)
- Standby mode for low power consumption
- 3Mbps high-speed Dynamixel bus for joint control
- Battery (30 minutes of operations), charger, and external power adapter (Battery can be removed from robot without shutting down by plugging in external power before removal)
- Versatile functionality (can accept legacy, current, and future peripherals)
- 3-axis gyro, 3-axis accelerometer, button x3, detection microphone x2

Appendix D. Tests with a real *DARwIn-OP* robot

A video showing one of the solutions that we ran in the actual robot can be found in https://www.youtube.com/watch?v=lOaWvOA9cb4. Figures 4(a) and 4(b) show the *DARwIn-OP* robot used in our experiments.

9. Docker: https://www.docker.com/



(*a*) Download mode, all motors disabled.

(*b*) Executing one of the experimental walking behaviors

Figure 4: Real *DARwIn-OP*

Appendix E. CMA-ES algorithm

While CMA-ES follows in essence the basic leitmotiv from Evolutionary Computation, it has the particularity that the amount of recombinants is fixed to one (a weighted mean of all the individuals of the current population), and this recombinant gives place to λ descendants through mutation. The next population is formed with the best μ children (deterministic selection and replacement). Also it has a great adaptation power to the fitness function

thanks to the update step of its covariance matrix. In Algorithm 2 we show how CMA-ES works.

input : define λ, μ
output: single individual
initialize m, σ_0, $C = I$, $p_\sigma = 0$, $p_c = 0$;
while *not terminate* **do**
  for $i \leftarrow 1, \lambda$ **do**
    $x_i =$ `SampleMultivariateNormal(`$mean = m$, $covariance\_matrix = \sigma^2 C$`)`;
    $f_i =$ `CalculateFitness(`x_i`)`;
  end
  $x_{i...\lambda} \leftarrow$ `Argsorted(`$f_{i...\lambda}$, $x_{i...\lambda}$`)`;
  $m' = m$;
  $m \leftarrow$ `UpdateMean(`μ,$x_1, \ldots, x_\lambda$`)`;
  $p_\sigma \leftarrow$ `UpdatePs(`p_σ, $\sigma^{-1} C^{-1/2}(m - m')$`)` ;
  $p_C \leftarrow$ `UpdatePs(`p_C, $\sigma^{-1}(m - m')$, $||p_\sigma||$`)` ;
  $C \leftarrow$ `UpdateCovariance(`C,p_c, $(x_1 - m')/\sigma, \ldots, (x_\lambda - m')/\sigma$`)` ;
  $\sigma_0 \leftarrow$ `UpdteSigma(`σ_0, $||p_\sigma||$`)` ;
end
return m *or* x_1 `// Either will do`

Algorithm 2: Pseudo-code of the CMA-ES algorithm

Appendix F. Execution results

The results of the experiments carried out for this project can be found in the tables below. For each combination of parameters, it shows the best fitness of the last generation, the total number of individuals that were generated before a stop condition was triggered and the number of individuals needed for obtaining one whose fitness is greater than 2.0.

It can be appreciated in Table 3 that some of the solutions found with high σ_0 values did not present the expected behavior. They correspond to robots that evolved a non-biped advancement technique, much like in the example shown in figure 5

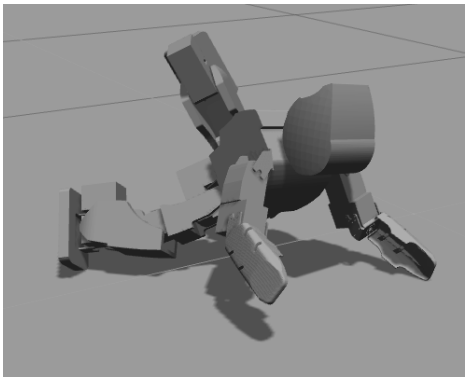


Figure 5: Non-biped advancement technique

σ_0	λ	μ	Best f	# Individuals	# Individuals for $f >= 2$
0.05	25	1	0.311	4050	N/A
0.05	25	5	0.765	10400	N/A
0.05	25	3	4.278	11050	2825
0.05	25	6	0.685	10250	N/A
0.05	25	12	0.765	9975	N/A
0.05	50	1	0.752	18400	N/A
0.05	50	5	3.167	24500	5150
0.05	50	6	0.618	17700	N/A
0.05	50	12	1.004	13950	N/A
0.05	50	25	1.003	21350	N/A
0.05	100	1	1.330	28800	N/A
0.05	100	5	0.623	23200	N/A
0.05	100	12	2.253	39500	11600
0.05	400	1	4.543	100000	18800
0.05	400	5	0.767	74400	N/A
0.05	400	50	6.682	100000	18800
0.05	400	100	5.500	100000	14400
0.05	400	200	6.457	100000	12000
0.05	800	5	4.039	100000	35200

Table 1: Values of *Max Fitness*, *Total Individuals*, *Individuals for Fitness* $>= 2$ for $\sigma = 0.05$

σ_0	λ	μ	Best f	# Individuals	# Individuals for $f >= 2$
0.10	25	1	2.541	10900	5950
0.10	25	5	2.730	9875	3025
0.10	25	3	0.673	10600	N/A
0.10	25	6	0.404	6075	N/A
0.10	25	12	0.595	5075	N/A
0.10	50	1	0.339	12300	N/A
0.10	50	5	1.072	15500	N/A
0.10	50	6	4.072	13200	3500
0.10	50	12	0.582	7700	N/A
0.10	50	25	2.680	46950	9950
0.10	100	1	3.045	32000	9500
0.10	100	5	4.110	33700	8900
0.10	100	12	1.161	25800	N/A
0.10	100	25	5.231	51700	5800
0.10	100	50	0.463	13700	N/A
0.10	200	1	5.053	62400	8200
0.10	400	1	4.864	100000	30400
0.10	400	5	4.382	97200	14400
0.10	400	50	5.477	100000	20000
0.10	400	100	0.491	52400	N/A
0.10	400	200	0.487	44800	N/A
0.10	800	5	6.284	100000	6400
0.10	800	50	5.699	100000	24800
0.10	800	100	0.424	55200	N/A
0.10	800	200	0.426	51200	N/A
0.10	800	400	0.424	52000	N/A

Table 2: Values of *Max Fitness*, *Total Individuals*, *Individuals for Fitness* $>= 2$ for $\sigma = 0.10$

σ_0	λ	μ	Best f	# Individuals	# Individuals for $f >= 2$
0.30	25	1	*3.049	8125	400
0.30	25	5	4.332	9050	1850
0.30	25	3	*0.363	3225	N/A
0.30	25	6	0.258	2975	N/A
0.30	25	12	0.351	5725	N/A
0.30	50	1	4.488	23100	3250
0.30	50	5	0.451	7400	N/A
0.30	50	6	*5.467	19200	2150
0.30	50	12	*4.210	19450	3000
0.30	50	25	*3.186	21950	4250
0.30	100	1	0.446	15000	N/A
0.30	100	5	0.511	16100	N/A
0.30	100	12	5.970	29000	3500
0.30	100	25	6.442	40000	7000
0.30	100	50	2.816	5400	4300
0.30	400	1	0.450	71600	N/A
0.30	400	5	0.491	68800	N/A
0.30	400	50	5.300	100000	13200
0.30	400	100	5.301	100000	15200
0.30	400	200	6.583	100000	23200
0.30	800	5	0.484	100000	N/A

Table 3: Values of *Max Fitness*, *Total Individuals*, *Individuals for Fitness* >= 2 for $\sigma_0 = 0.30$. Non-biped (crawling) individuals are marked with *