Worcester Polytechnic Institute
Digital WPI

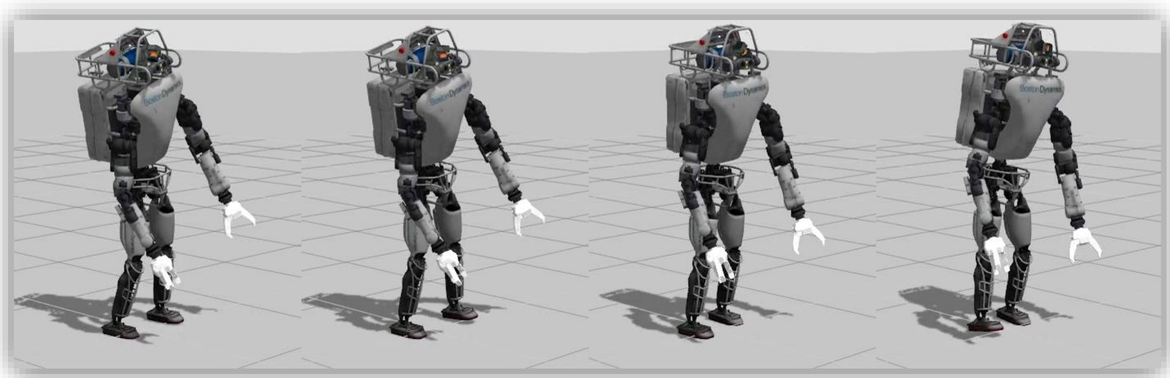
Major Qualifying Projects (All Years) Major Qualifying Projects

April 2016

Robot Learning

Batyrlan Nurbekov
Worcester Polytechnic Institute

Robot Learning



A Major Qualifying Project Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree in Bachelor of Science

By

Batyrlan Nurbekov

Michael A. Gennert, Project Advisor

April 25, 2016

Abstract

The purpose of this project is to mimic human learning and motion mechanisms in order to create an adaptive walking gait on a compliant humanoid robot - Atlas. This project applies the neural controller theory based on Central Pattern Generators (CPG) to reduce a state (parameter) space from 100 states to an average of 10 states. The goal of this learning mechanism is to find global optimal set of parameters for CPG while utilizing unsupervised learning based on self-organizing maps and rewards that adapt throughout the learning process. The learning mechanism also utilizes Covariance Matrix Adaptation – Evolutionary Strategies in order to converge to the parameter region that leads to a stable walking gait (success region) quickly. The experimental results demonstrate that the system is capable of learning how to make several steps over the course of 300 learning trials.

Acknowledgements

I express my deepest sense of gratitude and appreciation to my advisor, Dr. Michael Gennert. His help and guidance allowed me to successfully finish this work. Also, his support during the hardest moments of working on the project meant a lot to me.

I would like to thank my fellow students in WHRL who were always there to help me: Perry Franklin, Christopher Bove, Josh Graff, Lening Li, Aditya Bhat.

I would like to give credit to the former WPI – CMU team that participated in DARPA robotics challenge for developing code base for communications on the robot and basic functionality that was used in this project. I am also grateful to Kevin Knoedler who helped to figure out how communications on OCU network work.

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

There have been tremendous advances in solving the problem of generating a walking gait for humanoid robots using a controls approach (as it has been shown by DARPA robotics challenge). Even though, the control approach is robust it lacks the generalization needed for walking in a dynamic environment. It also lacks generalization in terms of other motor tasks (i.e. the controller implemented for walking frequently cannot be applied to the tasks of crawling, reaching or swimming).

As a solution to this problem a biologically inspired approach was proposed – using Central Pattern Generators (that reduce the dimensionality of the problem to make it feasible for the learning mechanisms to tackle the learning task) in combination with reinforcement learning. This approach allows the robot to adjust to environmental changes by using some form of learning (reinforcement learning) specific to the problem. It can also tackle other motor tasks by rewiring the connections between neurons.

There have been successful attempts of applying such systems to non-compliant humanoid robots (e.g. Nao, Hoap). However, no research has implemented (to date) the aforementioned mechanism for compliant humanoid robots, like Atlas. Therefore, this research is the first attempt of applying CPG, along with the new learning mechanism proposed in this paper to compliant humanoid robots.

2 BACKGROUND

2.1 CENTRAL PATTERN GENERATOR

Central Pattern Generators (CPGs) are neural networks that produce rhythmic patterns (e.g. oscillatory patterns, such as sine waves). They generally consist of three layers, as it is shown on the figure below.

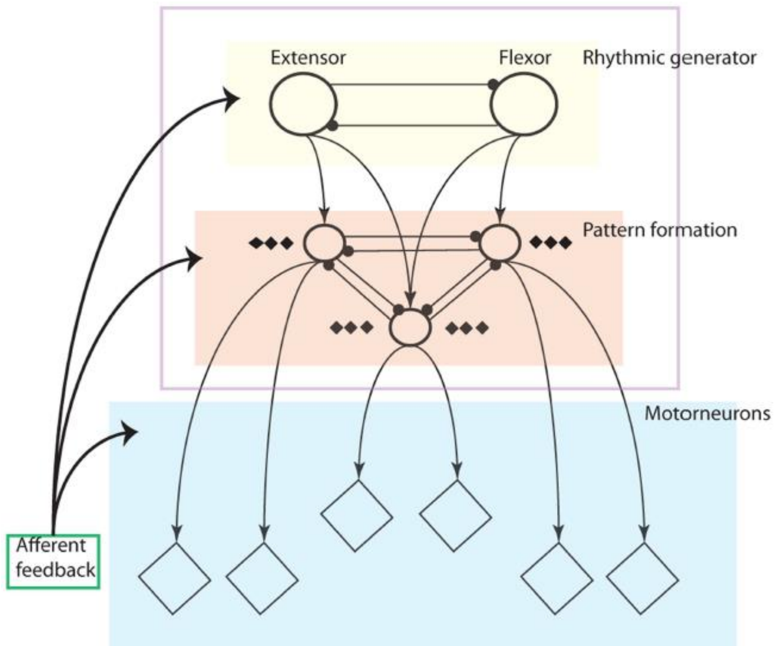


Figure 1. Layers of Central Pattern Generator (Li, Lowe and Ziemke)

2.1.1 Rhytmic generator layer.

The Rhytmic Generator (RG) layer performs functions of generating rhythmic patterns and synchronization of patterns between joints. The most frequent design has a pair of cells for each joint: extensor and flexor neurons. Rowat & Selverston proposed a model of a cell (neuron) for which two groups of currents are defined: a slow current and a fast current (Rowat and Selverston). These two types of currents are defined by the differential equations below:

$$Fast\ current: \tau_m \frac{dV}{dt} = -(Fast(V,\sigma))$$

$$Slow\ current: \tau_s \frac{dq}{dt} = -q + q_\infty(V)$$

Fast current function $Fast(V,\sigma)$) is a non-linear current-voltage function for the fast current. This function induces different behaviors (oscillating, damped oscillating and plateau potentials) for neurons in this model. It is defined in the equation below.

$$Fast(V,\sigma) = V - A_f \tanh(\frac{\sigma_f}{A_f} V)$$

The steady-state value of the slow current is proportional to V and σ_s, and is defined as (Amrollah and Henaff):

$$q_\infty(V) = V\sigma_s$$

2.1.2 Pattern formation layer

The role of the Pattern Formation (PF) layer is to modulate signals coming from RG layer based on afferent sensory feedback (both proprioceptive and exteroceptive). Each neuron in this layer is defined by sigmoid activation function:

$$PF_i = \frac{1}{1 + e^{-\alpha(I-\theta)}}$$

The incoming signal can be modulated by changing parameters α and θ above.

Also, I in the above equation is defined as the weighted sum of inputs:

$$I = \sum_{i=0}^{n} w_i I_i + \sum_{j=0}^{m} w_{RG\to PF,\, j} I_j$$

2.1.3 Motorneuron layer

The role of the Motor neuron (MN) layer is similar to the role of the PF layer. The difference between two layers is that the PF layer is generally supposed to modulate incoming signals in a more complex way, whereas MN layer directly integrates afferent feedback into the signal coming from the PF layer (to model reflexes).

Each neuron in MN layer is also defined as sigmoid activation function, processing the inputs as a weighted sum of incoming signals.

2.1.4 Sensory neurons

Sensory neurons are also modeled as Sigmoid Activation Function. There are several types of sensory neurons:

- Extensor (ES) and Flexor (FS) neurons. These neurons detect extension or flexion in a specific joint.
- Fall back (FB) and fall forward (FF) neurons detect when the robot is falling by monitoring the difference between the center of the support polygon and the projection of center of masses onto the ground plane.
- Anterior extensor (AS) neurons detect extreme hip angle which triggers knee extension reflex (knee becomes straight). This straight position of a knee is kept until the hip angle decreases (or reaches minimum value).

2.2 Self-Organizing Maps

2.2.1 Overview

Self-organizing map (SOM) is a type of a neural network that is trained using competitive learning, as opposed to error-correction learning, such as backpropagation. It creates a low-dimensional representation of the input space of the training samples which is often referred to as a map. This learning technique is usually considered to be a type of unsupervised learning and was introduced by Kohonen (Kohonen).

SOMs operate in two modes: training and mapping. The training mode builds the map using the inputs, whereas the mapping mode classifies the new input vector based on the already constructed map (during the training mode).

2.2.2 Algorithm description

The algorithm used for SOM's training is listed below (Wikipedia contributors).

Randomize the map's nodes' weight vectors;
Traverse each input vector in the input data set:
  Traverse each node in the map:
    Use Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector;
    Track the node that produces the smallest distance (this node is the best matching unit, BMU);
  Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector:

$$\boldsymbol{W}_v(s + 1) = \boldsymbol{W}_v(s) + \Theta(u, v, s)\, \alpha(s)(\boldsymbol{D}(t) - \boldsymbol{W}_v(s))$$

Increase s and repeat from step 2 while $s < \lambda$;

Where variables are defined as follows:

- S-current iteration
- λ-iteration limit
- t-index of the target input data vector in the input data set D
- $D(t)$-target input data vector
- v-index of the node in the map
- W_v-current weight vector
- u-index of the best matching unit (BMU) in the map
- $\Theta(u, v, s)$-restraint due to distance from BMU, usually called the neighborhood function
- $\alpha(s)$-learning restraint due to iteration progress

The illustration of the above algorithm is shown on the figure below. SOM (grid) converges to a specific region (blue). Nodes (within yellow circle) are pulled closer the sampled input vector. Over many iterations the map creates the representation of the region.

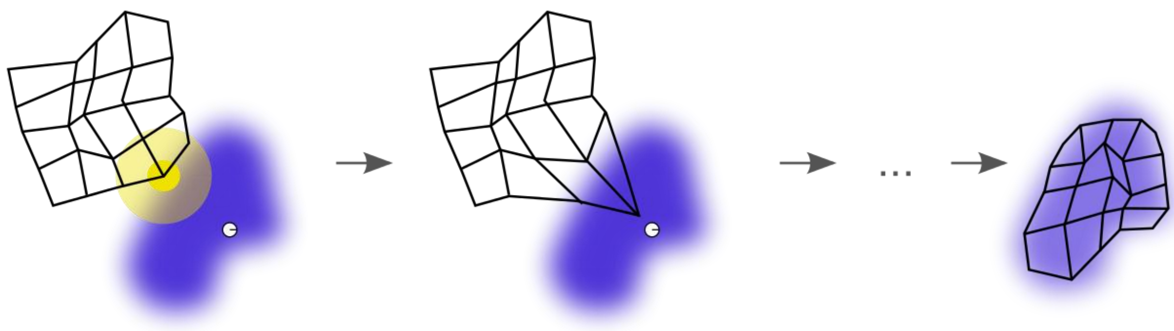


Figure 2. Illustration of SOM training algorithm

2.3 Qualitative Adaptive Reward Learning

2.3.1 Overview

The Qualitative Adaptive Reward Learning was first introduced by Nassour and is considered to be a reinforcement learning algorithm (direct policy search) that utilizes previous experience to make decisions (Nassour). The algorithm uses two SOMs (one to represent the success map and another to represent the failure map). In the original algorithm success map represents a region in the parameter space that led to a successful walk (the definition of successful walk can be varied based on the particular application). On the other hand, the failure map represents a region in the parameter space that led to the failure.

The entire algorithm is summarized in the flowchart below.

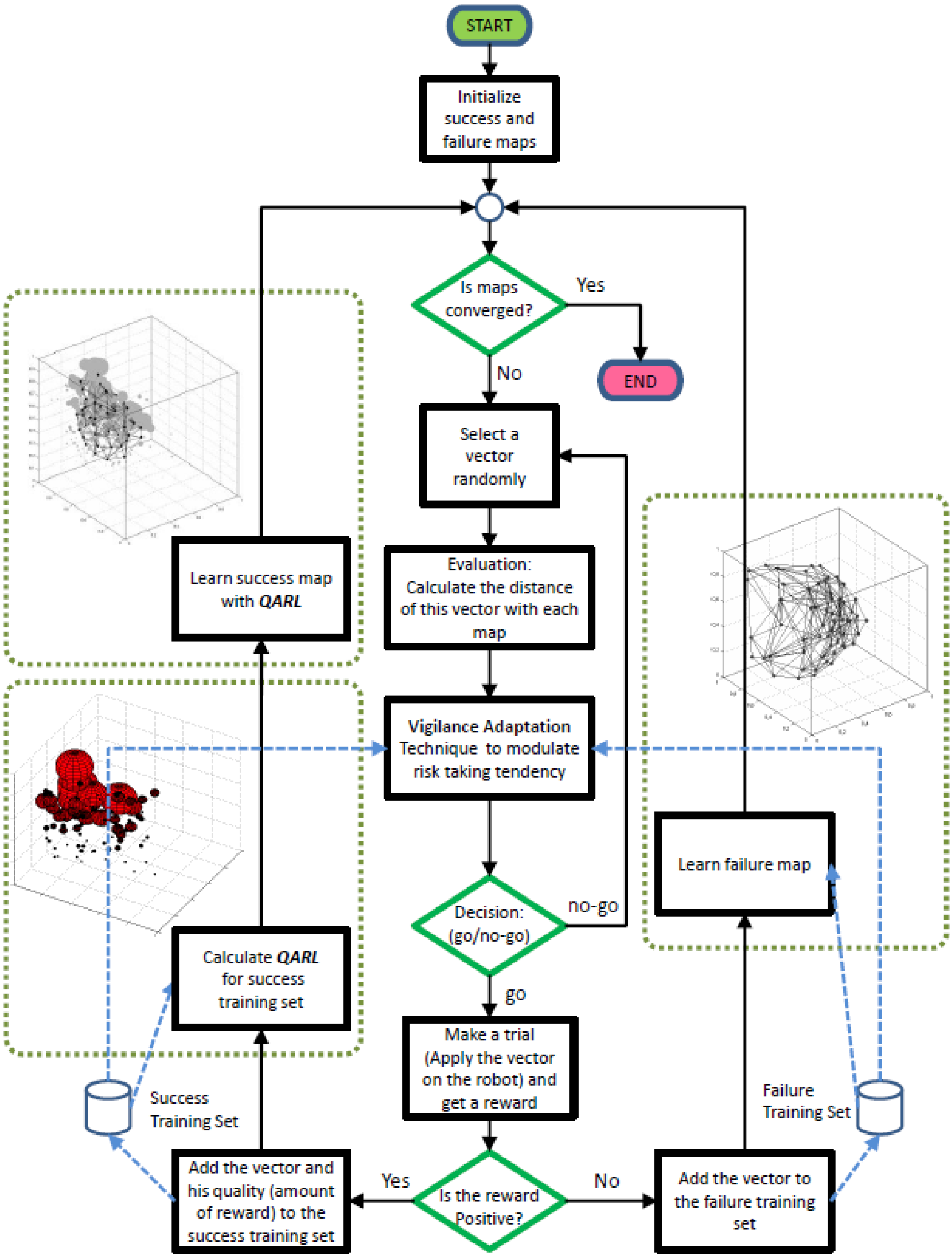


Figure 3. Qualitative Adaptive Reward Learning (Nassour)

The key aspect of this algorithm is that it can disregard newly sampled vectors, if the sampled vector might potentially lead to failure. In particular, the algorithm computes the distances from the sampled vector to the Best Matching Units (defined in the previous section) in success and failure maps, and compares the difference between the distances to the pre-defined threshold (so called Vigilance Threshold). If the distance is greater, then the algorithm runs learning iteration with that vector. Otherwise, it resamples another vector.

2.3.2 Reward Adaptation

Another convenient feature of this algorithm is the reward adaptation throughout the learning process. It is usually hard to establish the range limits (maximum and minimum) of the reward values experimentally (for normalization purposes) in the beginning of the training process. This feature allows to adapt the reward during the learning automatically by determining the range limits (η_{min} and η_{max}) of the reward after each trial.

The main idea is to add another multiplier term to the Weight Vector (defined in the previous section), $\rho(s)$. Therefore, the new weight vector would be defined as:

$$\boldsymbol{W_v}(s + 1) = \boldsymbol{W_v}(s) + \rho(s)\, \Theta(u, v, s)\, \alpha(s)(\boldsymbol{D}(t) - \boldsymbol{W_v}(s))$$

$$\rho(s) = \begin{cases} \rho_{\max}, & s = 0 \\ (\rho_{\max} - \rho_{\min}) * \dfrac{\eta(s) - \eta_{min}}{\eta_{max} - \eta_{min}} + \rho_{\min}, & s > 0 \end{cases}$$

$$\begin{cases} \eta(s) = Reward(v(s)) \\ \eta_{max} = \max(\eta(s = 0, \dots, S)) \\ \eta_{min} = \min(\eta(s = 0, \dots, S)) \end{cases}$$

In this case $Reward\big(v(s)\big)$ can be interpreted as the reward for the sampled input vector v at time step s. This reward function can be defined as any external criteria specific to the application (such as efficiency in terms of power consumption during when walking).

Lastly, reward adaptation can be introduced both for success and failure maps.

2.4 Covariance Matrix Adaptation – Evolutionary Strategies

2.4.1 Overview

Covariance Matrix Adaptation – Evolutionary Strategies (CMA-ES) is an evolutionary algorithm for non-linear non-convex black-box optimization problems in continuous domain. One of its features is that it works on a rugged search landscape (e.g. discontinuities, noise).

The algorithm samples new candidate solutions according to multivariate normal distribution (similar to other ES algorithms). Recombination is defined by selecting a new mean value for the distribution. Mutation is defined by adding a random vector (Wikipedia contributors).

The maximum-likelihood principle is exploited by the algorithm. It ensures that the mean of distribution is updated, such that the likelihood of previously successful solutions is maximized. Also, the covariance matrix (which specifies pairwise dependencies between the variables in the distribution) is updated, such that the likelihood of previously successful search step is maximized. Both updates are similar in nature to natural gradient descent.

The entire algorithm is listed below.

Set λ // number of samples per iteration, at least two, generally > 4
Initialize m, σ, $C = I$, $p_\sigma = 0$, $p_c = 0$, // initialize state variables
While *not terminate* // iterate
  For i **in** $\{1 \dots \lambda\}$ // sample λ new solutions and evaluate them
    $x_i = sample\_multivariate\_normal(mean = m, covariance\_matrix = \sigma^2 C)$
    $f_i = fitness(x_i)$
  $x_{1\dots\lambda} \leftarrow x_{s(1)\dots s(\lambda)}$ with $s(i) = argsort(f_{1\dots\lambda}, i)$ // sort solutions
  $m' = m$ // we need later $m - m'$ and $x_i - m'$
  $m \leftarrow update\_m(x_1 \dots x_\lambda)$ // move mean to better solutions
  $p_\sigma \leftarrow update\_ps(p_\sigma, \sigma^{-1} C^{-0.5}(m - m'))$ // update isotropic evolution path
  $p_c \leftarrow update\_pc(p_c, \sigma^{-1}(m - m'), ||p_\sigma||)$ // update anisotropic evolution path
  $C \leftarrow update\_C(C, p_c, \frac{x_1 - m'}{\sigma}, \dots, \frac{x_\lambda - m'}{\sigma})$ // update covariance matrix
  $\sigma \leftarrow update\_sigma(\sigma, ||p_\sigma||)$ // update step-size using isotropic path length

Return m or x_1

In addition, the illustration of operation of the algorithm over successive iterations is shown on the figure below.

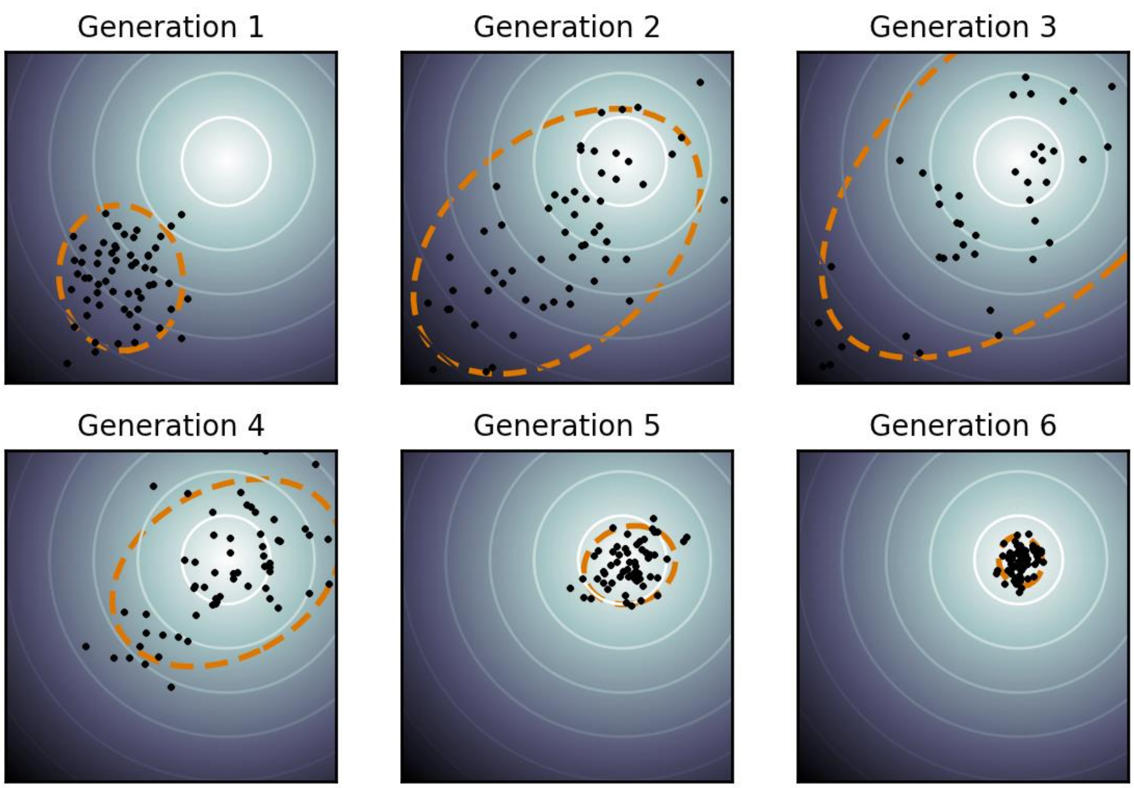


Figure 4. Illustration of CMA-ES algorithm

3 METHODOLOGY

3.1 SYSTEM OVERVIEW

The proposed system consists of three modules: Command Module, Learning Module, and Control Module.

The diagram of the system can be seen below.

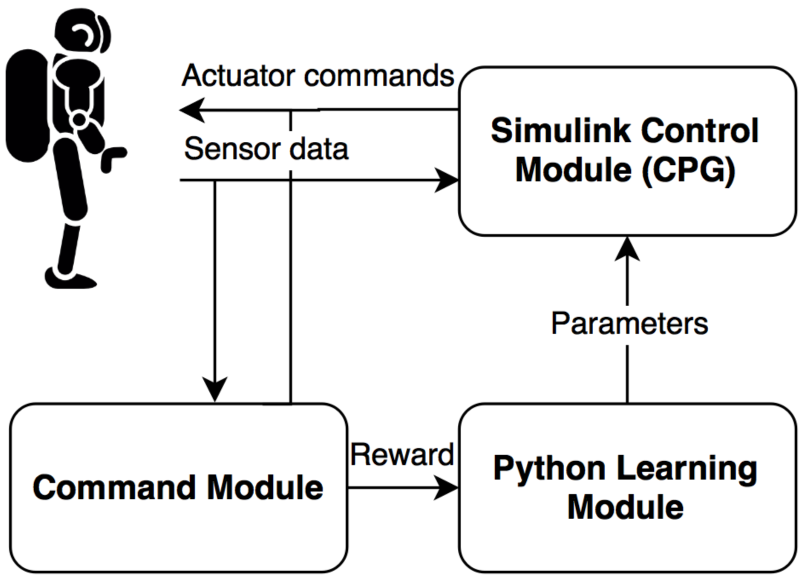


Figure 5. System overview

The primary function of the **command module** is to set up the trial by sending actuator commands and receiving sensor data. Its additional function is to compute a reward (defined in the next sections) by the end of the trial.

The **learning module** receives the rewards from the command module and performs necessary learning. It also outputs a new set of parameters that will be used during the next trial to the control module.

The **control module** receives a new set of parameters from the learning module and sets them up internally. It also runs in real-time during the trial by receiving sensor data from the robot and sending actuator commands.

ROS was used for communications between modules. Also, Matlab Python API was used for communications between the control and the learning modules.

3.2 Approach

The suggested approach to the problem is to use Gazebo simulator for finding an optimal set of parameters in simulation. After that, it is proposed to load the identified set onto the robot.

The advantage of this particular approach is that it allows to run large amount of trials initially. After uploading the set of identified parameters onto the robot it is possible to run learning again to make sure that the small differences in dynamics of the simulated and the real robot are accounted for.

3.3 Command Module

3.3.1 Implementation

In order to implement the command module Python language was used.

The logic of this module is simple and is demonstrated by the flowchart below.

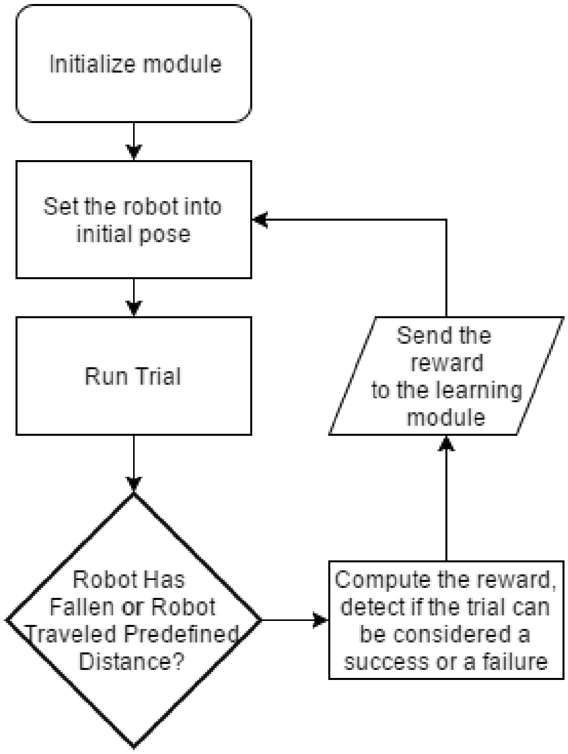


Figure 6. Logic of the command module.

3.4 CPG

3.4.1 Role in the system

CPG plays two major roles in the system:

- Reduction of a state (parameter) space from 100 states to an average of 10 states.
- Generation of rhythmic patterns.

Reduction of the parameter space (dimensionality) comes from the fact that patterns generated for each joint are automatically synchronized with each other. It also comes from the fact that a fixed oscillatory pattern is used instead of applying motion planning. In general, this amounts to reducing the problem to learning proper modulation of the output signal (i.e. amplitude, phase shift, frequency of the oscillatory pattern).

Generation of rhythmic patterns is the main property of CPG that sets the robot into motion.

3.4.2 Implementation

For the implementation of the control module MATLAB Simulink was used due to the fact that it allows for easier management of connections inside the system (CPG).

The implementation of the RG neurons in Simulink is shown on the figure below. It follows the equations given in the Background section.

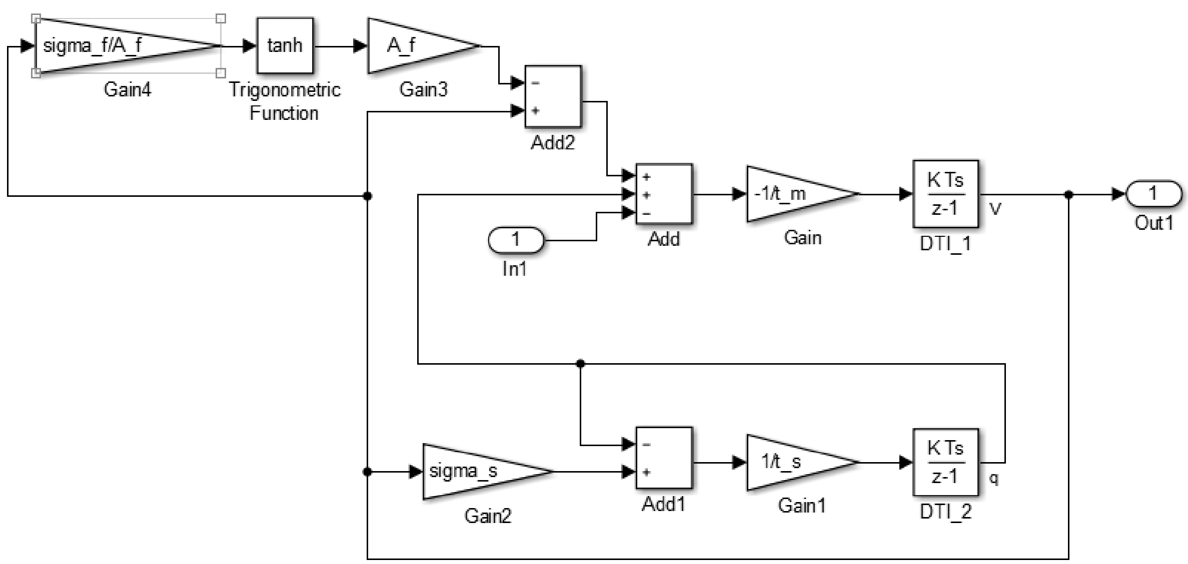


Figure 7. RG neuron implementation in MATLAB Simulink

The connection between RG neurons in different joints were implemented following suggestions in Nassour’s work.

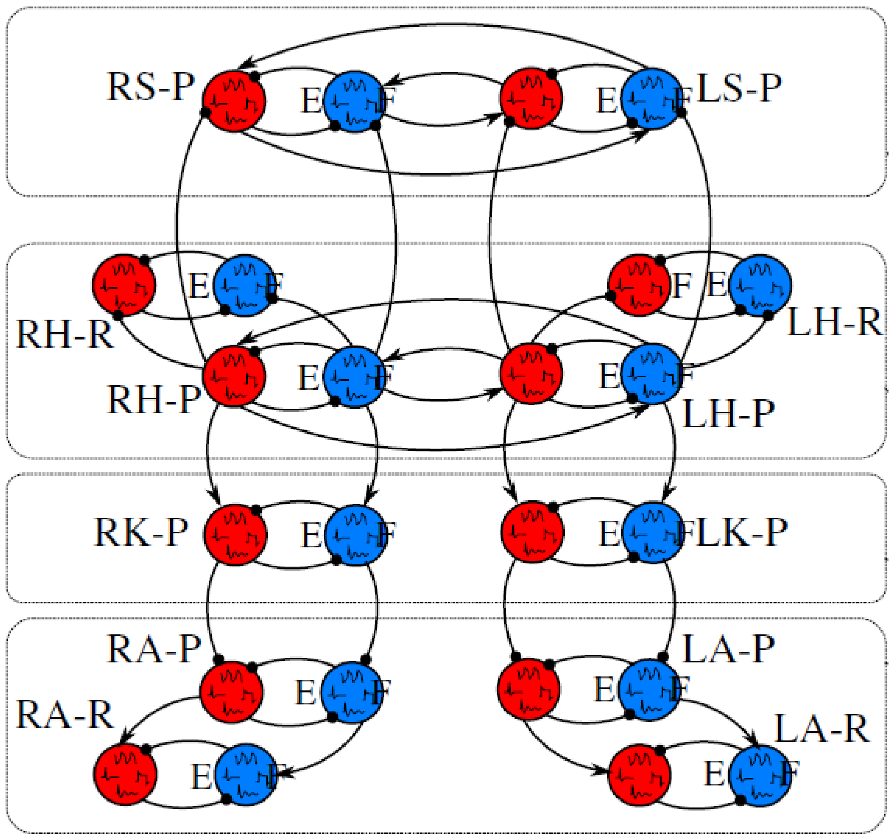


Figure 8. Connection between RG neurons in different joints (Nassour)

An example of PF and MN layer implementation for the ankle roll joint is shown below. FF and FB sensory signals are integrated in the PF layer, whereas ES and FS sensory signals are integrated in the MN layer.

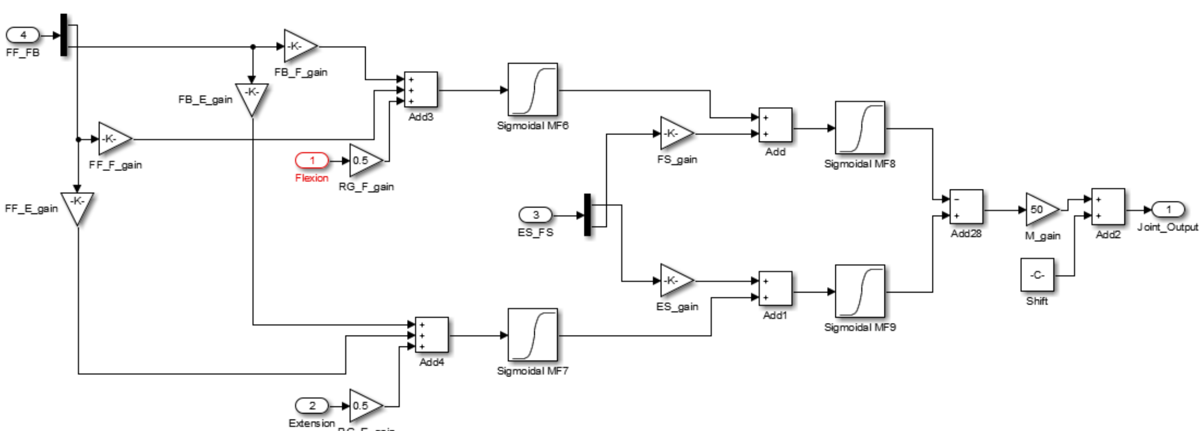


Figure 9. Pattern Formation and Motor Neuron layers for the ankle roll joint

ES and FS neurons were implemented using "sigmoid membership function" with "a" parameters having the opposite signs.

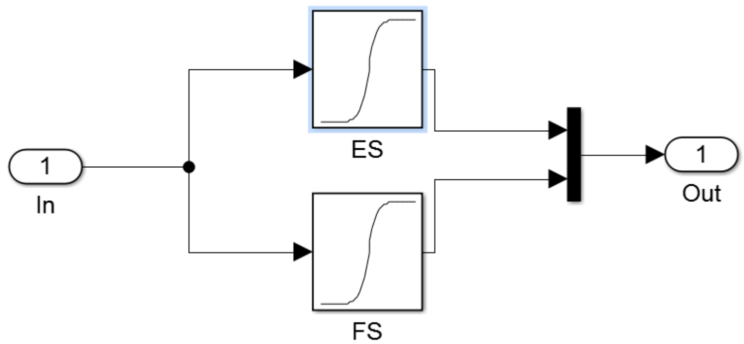


Figure 10. Implementation of ES and FS sensory neurons

3.5 Learning

3.5.1 Definitions

3.5.1.1 Success and Failure parameter regions

For our application the success region is defined as a region in parameter space (for CPG) that allows the robot to travel a pre-determined distance without falling.

The failure region is defined as a region in parameter space that leads the robot to falling during trial before reaching some pre-determined distance (defined as success).

3.5.1.2 Reward

There are two different types of reward: one for success region and another one for failure region.

The reward for failure region is defined as the time the robot was able to stay in the air without falling. The distance traveled was not taken into account for this reward. This was done because during the trials it was observed that including the distance traveled might lead the robot to be stuck in local minima (the robot swings its hip joints as much as possible to propel itself forward as far as possible). Also, the fact that CPG oscillates (it is not possible to shut off oscillations for all joints due to our selection of parameters to be optimized) leads the system to explore walking instead of standing still in place in order to remain stable longer.

The reward for success region is defined as the time the robot it took the robot to reach a pre-defined distance. In other words, the reward tracks how fast the robot was able to travel that distance. Another term can also be added to this reward that account for energy consumption during the walk.

3.5.1.3 Parameters learnt

In order to allow for faster convergence of the algorithm 4 parameters of CPG were selected for optimization: RG->PF weights for hip roll, hip pitch, knee, and ankle pitch joints.

3.5.2 Role in the system

The learning module consists uses two algorithms: CMA-ES and QARL.

3.5.2.1 CMA-ES

The primary purpose of this algorithm is to converge to the success region in parameter space. This addition to the original QARL algorithm accounts for the fact that the success region is narrow on compliant humanoid robots (that are highly unstable).

3.5.2.2 QARL

The purpose of this algorithm:

- Memorizing a particular region in parameter (state) space that leads to success/failure.
- Finding an optimal set of parameters in success region after failure region is learnt.

As CMA-ES algorithm samples new sets of parameters from failure region the failure map simultaneously learns. Each neuron in the failure map gets pulled to the sampled configuration by the distance defined by the failure map reward mentioned earlier.

Once CMA-ES converges (finds) to the success region it is completely turned off and pure QARL learning is run. It also might be reasonable to run some amount of successful trials with CMA-ES and learning the success map at the same, so that the closest neurons in the success maps converge to that region.

Summarizing, it can be noted that this algorithm plays the role of the memory in the system that learns a low-level representation of the input space. This allows to apply past experience in the future in order to learn the most efficient successful walking gait.

3.5.3 Implementation

In order to implement the learning module Python language was selected due to its extensive library support for applications that involve learning.

The high-level flowchart for the learning algorithm can be seen below.

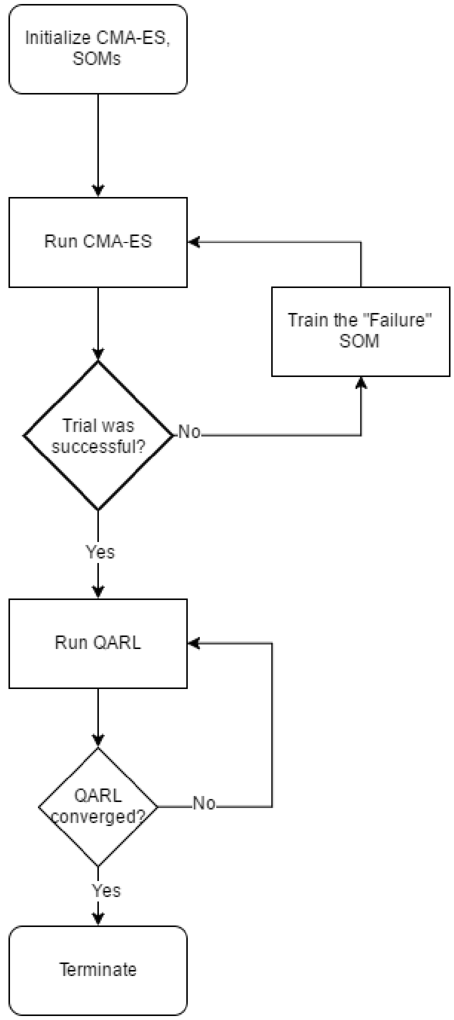


Figure 11. High-level flowchart of the learning system

4 RESULTS

The learning module was able to learn a policy (set of parameters) that led the robot to successfully making three steps in backward direction over the course of 300 trials.

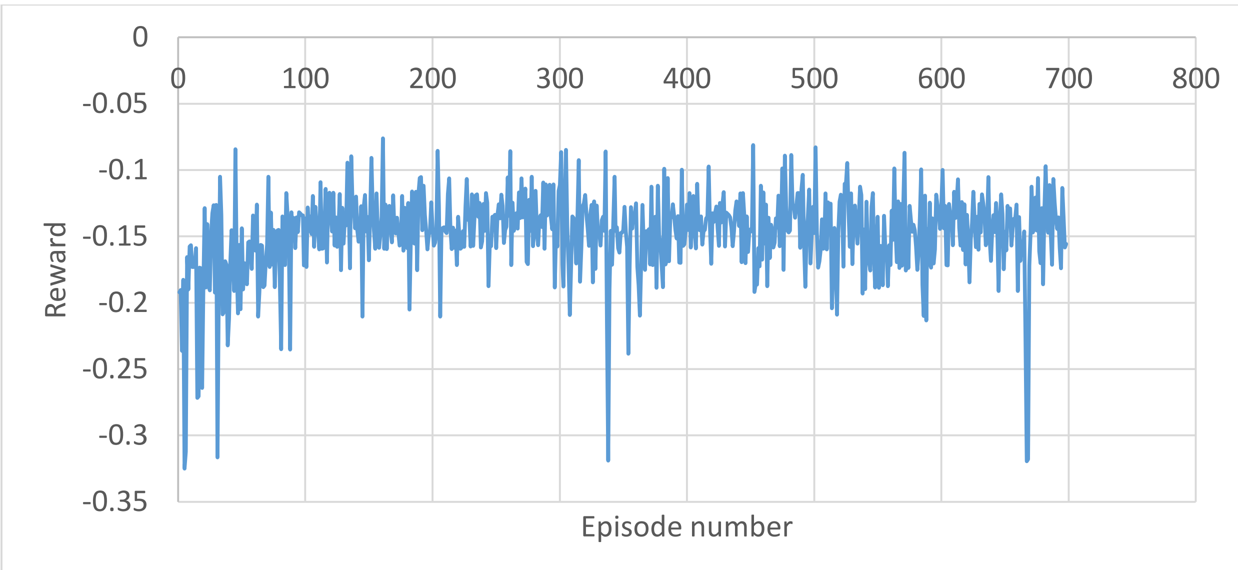


Figure 12. Reward over the course of 700 trials

From the figure above it is clear that the learning mechanism converges after the first 200 trials, and oscillates around the same reward (due to exploration).

Also, the sequence of images that shows Atlas making two steps is shown below.

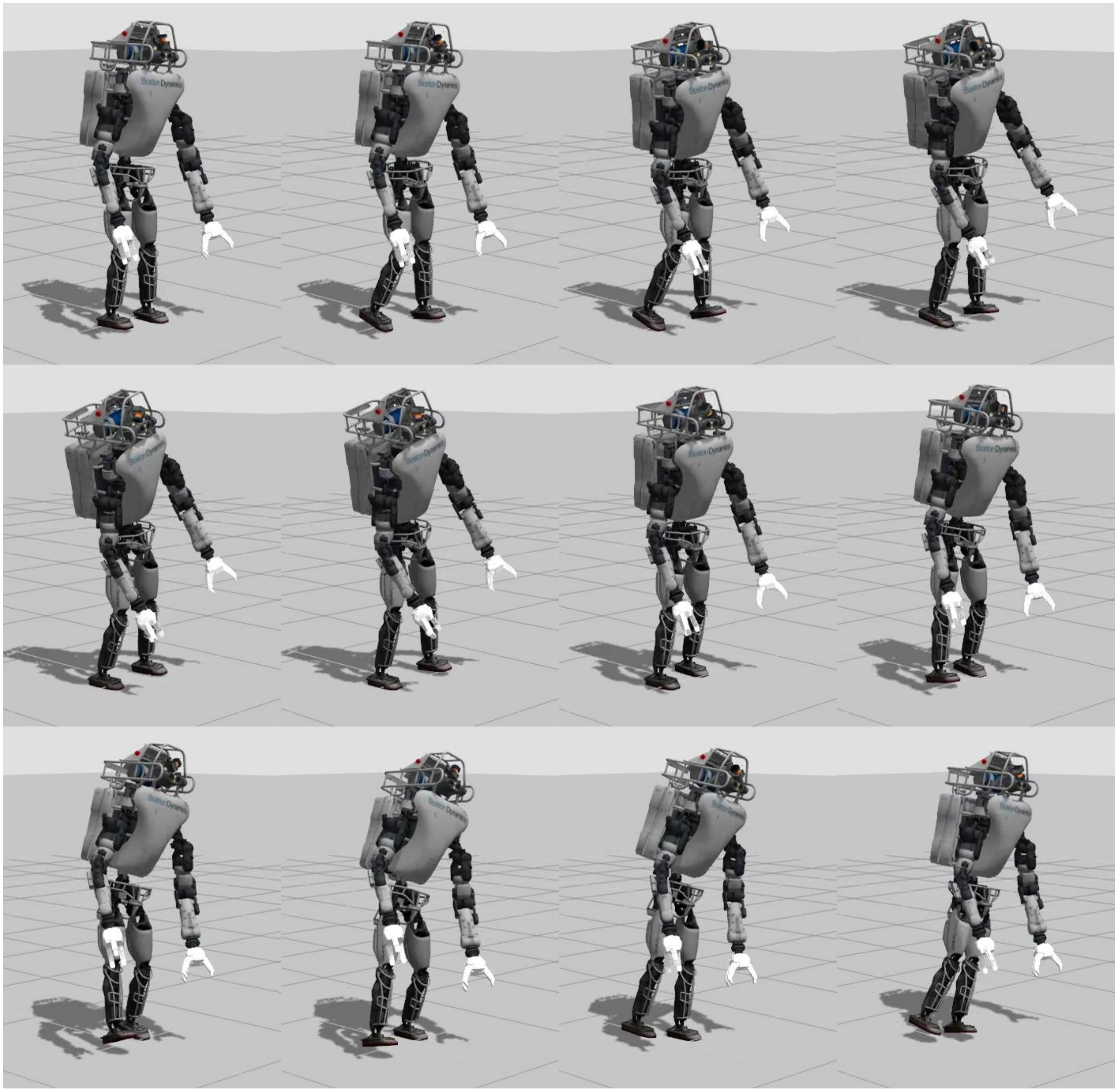


Figure 13. Robot making two steps in backward direction (animation starts at top left and ends at bottom right)

5 Conclusions and Future Work

5.1 Conclusion

A method for learning a biologically plausible walking gait on a compliant humanoid robot has been presented.

In addition, an approach of the dimensionality reduction for the problem has been shown.

5.2 Future Work

Overall, four major improvements are proposed:

- Generalization of the learnt and memorized experience to environmental changes (e. g. slopped floor, stairs).
- Automatic CPG controller structure construction (RG neuron re-wiring mechanism) to allow for other motor tasks, such as crawling and swimming.
- A balancing module that could:
    - account for Capture Point dynamics in order to make the gait more robust and allow the robot to walk without falling;
    - learn online and make adjustments to the robot's posture in real-time (NeoRL and CACLA).
- An improved learning mechanism that could account for multiple (separated) success regions.

6 REFERENCES

Amrollah, Elmira and Patrick Henaff. "On the Role of Sensory Feedbacks in Rowat–Selverston CPG to Improve Robot Legged Locomotion." *Frontiers in Neurorobotics 4* (2010): 113.

Kohonen, Teuvo. "Self-Organized Formation of Topologically Correct Feature Maps." *Biological Cybernetics 43* (1982): 59-69.

Li, Cai, Robert Lowe and Tom Ziemke. "Humanoid learning to walk: a natural CPG-Actor-Critic." 2013.

Nassour, John. "Success-Failure Learning for Humanoid: study on bipedal walking." PhD Thesis. 2013.

Rowat, P. and A. Selverston. "Learning algorithms for oscillatory networks with gap junctions and membrane currents." *Network: Computation in Neural Systems* (1991): 2(1):17–41. 4, 34, 63, 80, 82, 85, 121, 122.

Stowell, Dan. *Somtraining*.

Wikipedia contributors. *CMA-ES*. 23 March 2016. 24 April 2016.

—. *Self-organizing map*. 12 April 2016. 23 April 2016.