Evolution of adaptive behavior in a gravity varying environment

ACT Stage Report, ACT-TNT-AI-EABGVE

Advanced Concepts Team
European Space Agency

Nicolas Weiss

March 4 - July 31, 2009

Contents

Introduction 2
First section: PaGMO 3
    Introduction 3
    Work description 3
Second section: BOINC 7
    Introduction 7
    Work description 7
Third section: TwoDee 8
    Introduction 8
    Work description 8
Conclusion 48
Appendix A 49

Introduction

The activities of this stage have been carried out at the Advanced Concepts Team (ACT) at the European Space Research and Technology Centre (ESTEC) of the European Space Agency (ESA) located in Noordwijk, The Netherlands from March 4 until July 31 2009 comprising a total of 13 weeks without any leave or absent days. These were supervised by Dr. Christos Ampatzis and Dr. Dario Izzo.

The structure of this report reflects the course of activities developed during the stage. In the first section we will describe the work in the further development of a parallel global multi-objective optimizer (PaGMO) originally developed within the ACT. In the second section we will report on the first steps towards the setup of a distributed computing environment based on the BOINC infrastructure. The last section of this report will account for the main activity of this stage this being the development of a simulated environment for neurocontroller evolution in space relevant applications. We will finalize this report in the form of a conclusion.

First section: PaGMO

Introduction

Parallel Global Multi-objective Optimizer (PaGMO) is a massively parallel computing environment for global optimization in C++. It is based on the island model paradigm of parallel evolutionary computations and pushes this coarse grained approach for parallelization to global optimization in general. The asynchronous island-model framework is implemented. Objects such as island, archipelagi, individuals and populations can be instantiated as well as the available algorithms and available problems. The optimization on each island is assigned to a different thread of the underlying operating system so that parallelization is achieved automatically if multiple processors are available. New algorithms and problems can be easily added and tested within this framework. The main features of the code are exposed in Python to achieve an extremely user-friendly environment.

Work description

The work with PaGMO was carried out in the first weeks of the stage. The first step was to get familiarized with the exposure of PaGMO in Python and in order to achieve this, works were conducted on the reproduction of some of the results published in [8] which were produced with the original C++ implementation of the code. The test were performed with a 300-dimensional Schwefel Problem, a maximum number of function evaluations of 2 million and a threshold of 0,1. It is important to consider that the number of function evaluations in the parallel strategies have to be multiplied by the number of processors to get the proper value.

Algorithm	Trials	Success	Evaluations	Fitness
DE1	20	0	xx	236,88
DE2	20	14	230000	0,041
DE3	20	0	xx	467,77
DE4	20	19	380000	0,059
DE5	20	20	410000	0,059

Table 1: Sequential strategy with the following parameters: max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

Table 1, 2 and 3 show the results of the performed tests. We found a discrepancy with the results of the third strategy of Differential Evolution (DE3) for the parallel strategies. After debugging and finding the errors the tests were run a second time and again compared with the published results. This time having statistically speaking an exact match. A further test was performed in the second run. The sequential strategy was not only tested in an island but also in an archipelago with only one island. The reason for this being that an archipelago and an island are instantiated with different constructors. The results can be found in table 4, 5, 6 and 7.

The second part of the work with PaGMO was to create and prepare a Wiki (`http://pagmo.sf.net`) for the first public release of the code on SourceForge.

Algorithm	Trials	Success	Evaluations	Fitness
5xDE1	20	0	xx	118,44
5xDE2	20	20	220000	0,036
5xDE3	20	0	xx	267,26
5xDE4	20	20	380000	0,057
5xDE5	20	20	380000	0,058
5xDE1...DE5	20	20	260000	0,032

Table 2: Parallel random replacement strategy with the following parameters max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

Algorithm	Trials	Success	Evaluations	Fitness
5xDE1	20	19	180000	0,04
5xDE2	20	19	180000	0,039
5xDE3	20	0	xx	947,51
5xDE4	20	20	310000	0,055
5xDE5	20	20	330000	0,058
5xDE1...DE5	20	20	18000	0,029

Table 3: Parallel best/worst replacement strategy with the following parameters max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

Furthermore a tutorial for Python users was written. A Print Screen of the Wiki page can be found in figure 1 and one of the tutorial in figure 2.

The third part of the work with PaGMO has been the development of a fully automated Test Class to be released with the original code on SourceForge. The Test Class has been written in Python and allows the user to perform tests with different problems, algorithms, parameters, strategies, etc. and also to compare the results of different tests via a Welch's Test. The code can be found in Appendix A.

Algorithm	Trials	Success	Evaluations	Fitness
DE1	20	0	xx	236,88
DE2	20	11	230000	0,04
DE3	20	0	xx	630,00
DE4	20	19	370000	0,06
DE5	20	20	410000	0,06

Table 4: Sequential (isl) strategy with the following parameters max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

Algorithm	Trials	Success	Evaluations	Fitness
DE1	20	0	xx	118,44
DE2	20	20	220000	0,04
DE3	20	0	xx	592,69
DE4	20	20	380000	0,06
DE5	20	20	410000	0,06

Table 5: Sequential (archi) strategy with the following parameters max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

Algorithm	Trials	Success	Evaluations	Fitness
5xDE1	20	20	190000	0,048
5xDE2	20	20	220000	0,039
5xDE3	20	20	550000	0,0088
5xDE4	20	20	370000	0,0598
5xDE5	20	20	390000	0,057
5xDE1...DE5	20	20	190000	0,0433

Table 6: Parallel random replacement strategy with the following parameters max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

Algorithm	Trials	Success	Evaluations	Fitness
5xDE1	20	20	190000	0,033
5xDE2	20	20	220000	0,052
5xDE3	20	19	530000	0,0055
5xDE4	20	20	360000	0,0565
5xDE5	20	20	380000	0,0595
5xDE1...DE5	20	20	190000	0,0347

Table 7: Parallel best/worst replacement strategy with the following parameters max. function evaluations = 2.000.000, threshold = 0,1, problem = Schwefel(300)

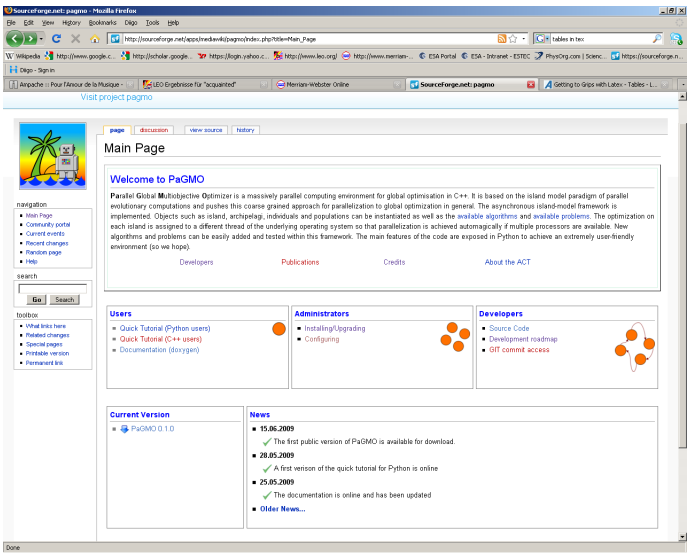


Figure 1: PaGMO's Wiki on SourceForge: `http://pagmo.sf.net`

Figure 2: PaGMO's quick tutorial for Python: `http://sourceforge.net/apps/mediawiki/pagmo/index.php?title=Quick_Tutorial_(Python)`

Second section: BOINC

Introduction

Berkeley Open Infrastructure for Network Computing (BOINC) is a non-commercial middleware system for volunteer and grid computing. It was originally developed to support the SETI@home project before it became useful as a platform for other distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics. The intent of BOINC is to make it possible for researchers to tap into the enormous processing power of personal computers around the world.

Work description

A distributed computing environment presents enormous benefits that allow researchers to reduce the calculation time of very large amounts of data. This case presents itself also in the area of evolutionary robotics. The evolution of complex tasks or behaviors are extremely time demanding (for some examples see [3]). For this reason the idea was born within the ACT to start up a BOINC infrastructure, and so be able once complex tasks to be evolved are determined, to distribute the evolution of the neurocontrollers among as many computers as possible, accumulating in this way a huge amount of computational power.

The main activities in this project were in the area of literature research and the setup of a BOINC server on a virtual machine. With the latter we were able to make first experiences with BOINC and identify the main aspects to be further developed. The works in this area were continued by another stagiere within the ACT and I focused on the further development of TwoDee and the setup of new tasks to be evolved. Both aspects will be described in detail in the next section.

Third section: TwoDee

Introduction

Evolutionary Robotics (ER) is a methodology that uses evolutionary computation to develop controllers for autonomous robots. Algorithms in ER frequently operate on populations of candidate controllers, initially selected from some distribution. This population is then repeatedly modified according to a fitness function. In the case of genetic algorithms (GA), a common method in evolutionary computation, the population of candidate controllers is repeatedly grown according to crossover, mutation and other GA operators and then culled according to the fitness function. In this particular case the controllers used are artificial neural networks (Perceptron, Multilayer Perceptron and Continuous Time Recurrent Neural Network (CTRNN)). The weights, biases and time constants (in the case of the CTRNN) are coded into a string of numbers called chromosomes. After every generation the behavior of the robot with a particular chromosome (i.e. controller) is evaluated with a fitness function that describes the task or behavior that is supposed to be evolved. The theoretical basis for this project will not be covered in this report, we recommend for the interested reader the following literature: for an introduction on Artificial Neural Networks [7], for a very theoretical approach to dynamical Neural Networks [4] and for some applications [3] and [13].

Work description

TwoDee is a two dimensional simulator with a custom rigid body engine, specialized to handle only the dynamics for a robot on flat terrain with holes. The code is written in object-oriented C++ in Hungarian notation. It compiles and runs on POSIX.1 compatible operating systems such as Linux, and with relatively few change it should be compilable and runnable on other operating system. The goal of the simulator is to simulate a virtual world and the interaction between a number of robots and static objects in an arena. Control programs running on physical robots often operate in discrete cycles. In each cycle sensory inputs (e.g. light sensors) are read, the controller decides how to act and sets the actuators (e.g. wheels) accordingly. Like many other simulators, TwoDee follows this discrete sense-act-update pattern as an approximation of continuous time. The simulation cycle is as follows, first the virtual world is setup, then a cycle is entered where first the sensor readings are computed and fed to the robot controllers which in turn determine the actions of the robot. The state and time of the virtual world is the updated by moving objects according to the forces currently acting on them. It is assumed that forces are constant between update cycles. Finally, the state of the virtual world can be visualized if the user is running the simulator in interactive mode. Typically robots run a control cycle every 0.1 seconds, thus per default one simulation cycle corresponds to 0.1 seconds of virtual time but the control frequency can be adjusted.

The first step in this project was to get familiarized with the C++ code of TwoDee which was mainly developed by A. Christensen [6] and C. Ampatzis [3] at the Universite Libre de Bruxelles. After identifying the basic structure of the code all our efforts were set to perform tests with different controllers and fitness functions and develop a complex task to be distributed when the BOINC

environment is ready. The experiments that we performed grow in complexity. We started with a very simple arena and a phototaxis task and ended with a changing gravity environment and fuel consumption minimization. During these experiments several fitness functions were tested, simple phototaxis (equation 7), phototaxis with speed minimization (equation 8) and phototaxis with speed minimization (equation 11). Towards the end of our work we implemented a thrusters model with tidal gravity in equations 12 to 17 that we then tested with different gravity environments.

For the first experiments of our work we used the in TwoDee already implemented model of a robot. Towards the end we developed a model of a spacecraft with thrusters under the influence of tidal gravity. The equations that characterise the first model can be found in equations 1 to 6.

$$x_{\mathrm{i}+1} = x_{\mathrm{i}} + \dot{x}_{\mathrm{i}} \cdot dt \tag{1}$$

$$y_{\mathrm{i}+1} = y_{\mathrm{i}} + \dot{y}_{\mathrm{i}} \cdot dt \tag{2}$$

$$\theta_{\mathrm{i}+1} = \theta_{\mathrm{i}} + \dot{\theta}_{\mathrm{i}} \cdot dt \tag{3}$$

$$\dot{x}_{\mathrm{i}+1} = \dot{x}_{\mathrm{i}} + a_{\mathrm{x}} \cdot dt \tag{4}$$

$$\dot{y}_{\mathrm{i}+1} = \dot{y}_{\mathrm{i}} + a_{\mathrm{y}} \cdot dt \tag{5}$$

$$\dot{\Theta}_{\mathrm{i}+1} = \frac{WheelSpeed_{\mathrm{R}} - WheelSpeed_{\mathrm{L}}}{Radius_{\mathrm{Chassis}} \cdot \pi} \tag{6}$$

A phototaxis task means for us in this context basically that the robot gets initialized on an arena with dimensions 10X10 where a light source is placed and the task that the robot has to perform is getting to the light. The fitness function that describes this behavior is very simple and can be found in equation 7.

$$f(t) = \frac{d_0 - d_{\mathrm{s}}}{d_0} \tag{7}$$

where d_0 is the initial distance and d_{s} is the shortest distance to the light.

As we expected the evolution of this behavior was done very quickly and with no apparent difficulties, after a few generations the fitness score converged to 97% of the maximum value. The trajectory plot for one evolved individual can be seen in figure 3. After the first few results we started to think about ways of complicating this particular task.

At this point a very interesting and uncommon space application was drawn to our attention by Dario Izzo:

Place a robot in an environment with gravity. Give him the ability to jump. Set as fitness the distance covered at the end of a simulation. Study under what gravity conditions a walking behavior is evolved or a hopping behavior is evolved. Clearly at low gravity regimes it is convenient to hop and at high gravity regimes it is convenient to walk. The critical value of gravity at which the transition

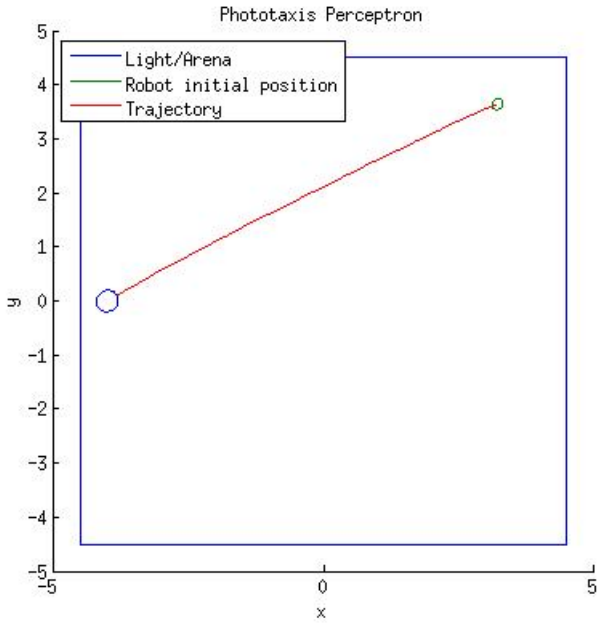


Figure 3: Simple phototaxis executed successfully by a perceptron

takes place depends on the robot morphology. So the question becomes: given the gravity is it better to design hoppers or walkers?

Starting from this idea we began to develop the first steps towards answering this question. The first matter to investigate was if it is possible to imagine a robot, i.e. a controller, that is able to find a general strategy that solves a given problem in a gravity variant environment. A general strategy because we are working with off line learning, that being the evolution is run and then the controllers are evaluated, i.e. when the evolution is over and the robot is placed in a new gravity environment it can not learn from his actions in real time, that is the reason why it has to evolve a general strategy that allows him to get to the light in every single environment (this is of course very difficult to evolve and is also the reason why similar environments are used for evolution and for evaluation). In the next subsections we will describe the different setups and experiments.

Experiment 1: Perceptron under fix gravity

Since we are working with a 2D simulator, the first simplification of the problem is going from a three dimensional problem to a two dimensional one. The robots will be placed in a squared arena with a light source analog to figure 3. At this point it is important to address the fact that the robot is not able to directly percieve drift in the environment. It only has 8 light sensors that allows it to percieve its relative position and orientation to the light source. The first setup is as follows: we represent gravity as a drift, we can imagine it as if the robots were experiencing that the floor is moving in a given direction. The controller

used is a perceptron and the right half of the arena has no drift (i.e. the floor is not moving) and the second one has drift pointing north (i.e. if the robot does not do anything it will drift towards the upper part of the arena). The place where this change in the conditions takes place is fixed to $x = 0$. This constrain could generate discontinuity issues, which we will disregard for the time being. The evolution was performed with a genetic algorithm with the following parameters: mutation rate = 0.1, number of crossovers = 1, number of elites = 1, crossover distance = 1, population = 50, generations = 50 and evaluation time = 50 s.

In order to allow us to visualize the change in the adopted strategies and if there is some kind of change in the evolved controllers, we placed not only the in this environment evolved individual but also an individual that was evolved in a no gravity environment (analog to figure 3). The trajectory plots can be found in figure 5.

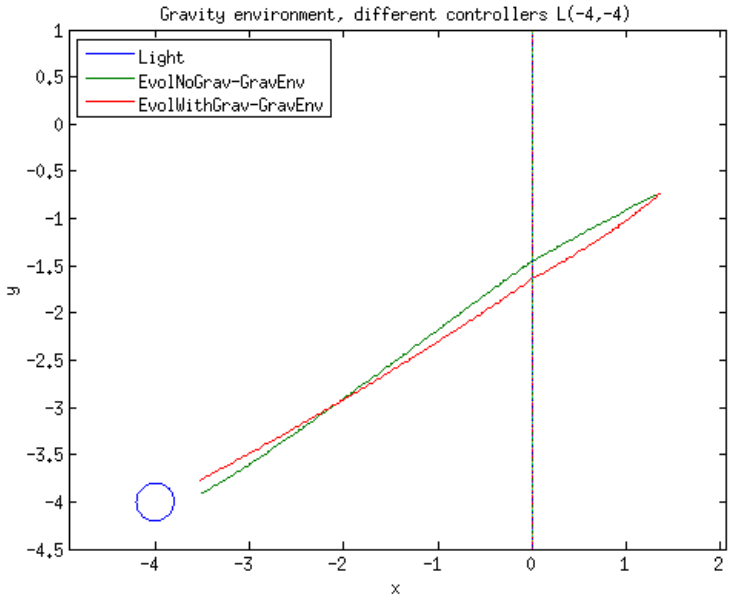


Figure 4: Simple phototaxis executed successfully by a perceptron evolved in a half gravity half no gravity environment and a perceptron evolved in a no gravity environment, both evaluated in a half gravity-half no gravity environment

As we can see in figure 5 the controller evolved in the gravity environment solves the task with almost the optimal trajectory (red straight line) and the controller evolved in a no gravity environment and evaluated in the half-half environment produces a longer trajectory (green curved line).

With the same setup we performed an analog test with the only difference being the environment, in this case both controllers are evaluated in a no gravity environment. The trajectory plots can be seen in figure 6. In this case as it was expected the controller evolved in a no gravity environment (green line) produces an almost optimal trajectory (straight line).

Since we are going to increase the complexity of the arena in the future

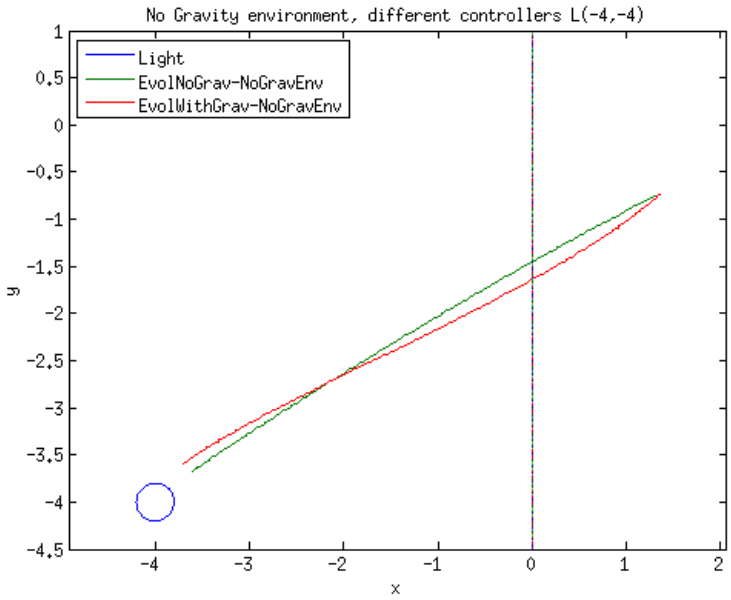


Figure 5: Simple phototaxis executed successfully by a perceptron evolved in a half gravity half no gravity environment and a perceptron evolved in a no gravity environment, both evaluated in a no gravity environment

we want to test this setup with more complex controllers, for this reason we will perform the same test with a multilayer perceptron and a continuous time recurrent neural network.

Experiment 2: Multilayer Perceptron under fix gravity

The setup of the arena and the parameters of the genetic algorithm are the same as in the first experiment. Again the comparison is performed between a controller evolved in a half gravity-half no gravity environment and a controller evolved in a no gravity environment, both being evaluated this time in a gravity environment. A multilayer perceptron with two hidden neurons was used as a controller. The trajectory plots for this experiment can be seen in figure 6

In the case of the multilayer perceptron as a controller the difference between the evolution taking place in a gravity and in a no gravity environment is much more obvious as in the case of the perceptron. This difference was to be expected since we increased the magnitude of the drift for this experiment in order to be able to observe this phenomenon more clearly. As in the first experiment the controller evolved in the half gravity-half no gravity environment solved the task better than the controller evolved in a no gravity environment.

The next experiment will be perform with an even more powerful controller, a continuous time recurrent neural network.

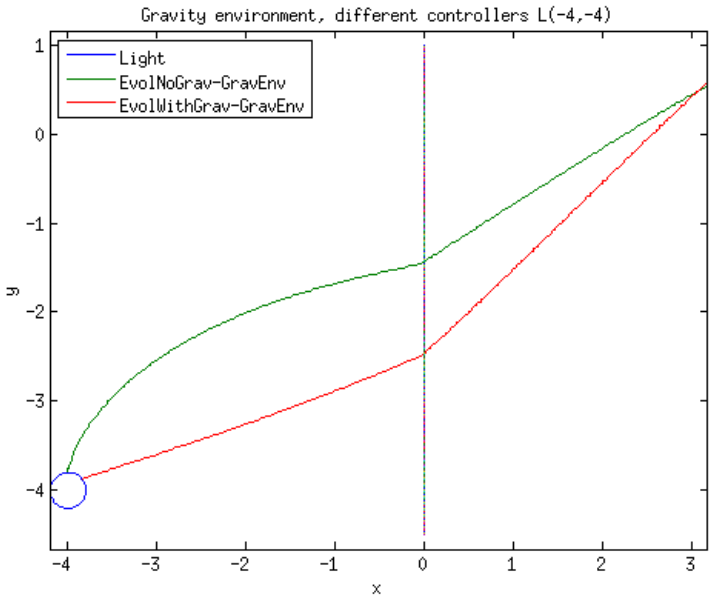


Figure 6: Simple phototaxis executed successfully by a multilayer perceptron evolved in a half gravity half no gravity environment and a multilayer perceptron evolved in a no gravity environment, both evaluated in a gravity environment

Experiment 3: Continuous time recurrent neural network under fix gravity

Since we are going to be using feed forward CTRNNs as a control structure for the rest of the experiments we will begin this section with a short description taken from [3].

Continuous Time Recurrent Neural Networks (CTRNNs) have been introduced in evolutionary robotics in [4]. They are the reflection of a theoretical approach to cognition which aims to exploit the mathematical tools of dynamical systems theory to investigate issues of interest in adaptive behavior research. We can say that an agent exhibits adaptive behavior when it is able to modify its behavior with respect to changing environmental conditions and/or changes in the behavior of other agents. According to Beer, there are two fundamental principles which justify the use of the formalism of dynamical systems theory within the context of adaptive behavior. First, since the fundamental nature of adaptive behavior in natural systems is to generate the appropriate behavior at the appropriate time, dynamical systems theory provides the required mathematical formalisms for the description and the analysis of systems whose behavior unfolds over time. For an embodied agent, time can make the difference between an adaptive behavior and an unsuccessful one. Second, it looks plausible to consider adaptive behavior as generated by causal mechanisms which result from the dynamical interactions of elementary units such as cells or molecules, rather than generated by the dynamics of the single elementary units. Thus, in an effort to explain adaptive behavior, we need to resort to the structure of the

internal dynamics; dynamical systems theory is the right tool to give us insight into such processes.

CTRNNs represent a convenient way of instantiating a dynamical system to control the behavior of autonomous robots. CTRNNs differ from the classical artificial neural networks (e.g., the perceptron), as they can be expressed as a system of differential equations. Each node within a CTRNN has its own state the activation level whose rate of change is specified by a time constant associated with each node. Furthermore, the nodes within the network are self-connected, as well as interconnected in an arbitrary way with each other. These two features allow the network to develop dynamical behavior in which the state of nodes alters the behavioral output of the system even if the sensory input remains constant.

CTRNNs are arguably the simplest nonlinear, continuous dynamical neural network model [4]; however, despite their simplicity, they are universal dynamics approximators in the sense that, for any finite interval of time, CTRNNs can approximate any smooth dynamical system arbitrarily well. Finally, fixed weight CTRNNs have been demonstrated to be able to produce learning behavior in robots; this goes against the belief that learning in ANNs necessarily corresponds to weight changes (plastic weights).

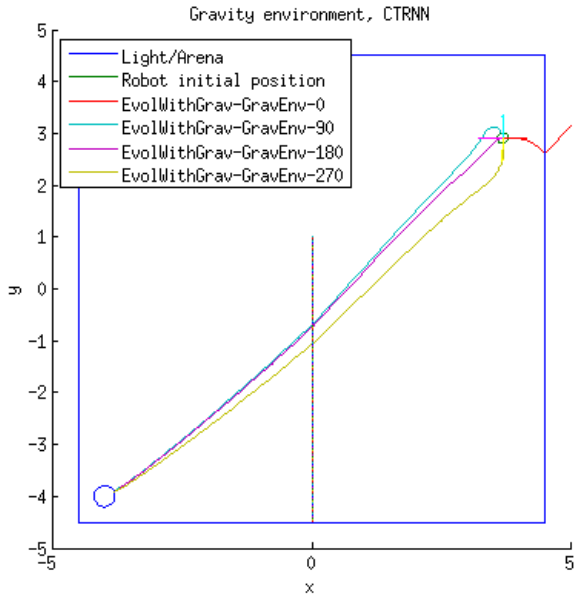


Figure 7: Simple phototaxis executed successfully by a CTRNN evolved in a half gravity half no gravity environment and evaluated in a gravity environment

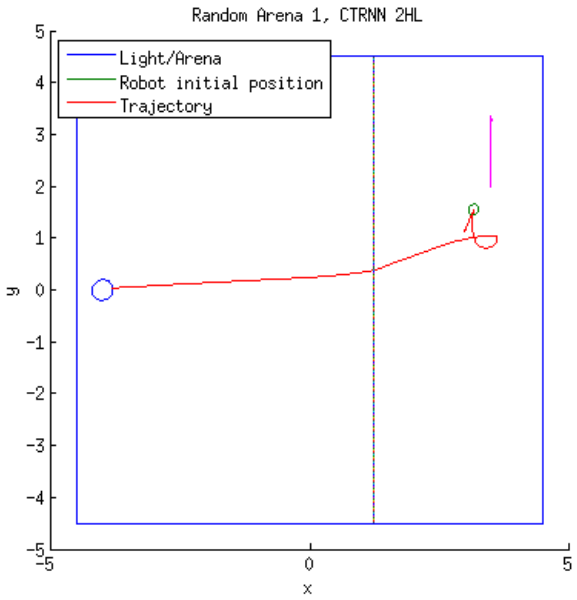
In figure 7 we can see the trajectory plots for an evolved CTRNN for different initial orientations. As we can observe the task is successfully solved in almost every case besides the case with an initial orientation of 0° (red line), this is due to the fact that the robot is initialized very close to the border of the arena and this distance is not enough to complete the turn to face the light.

Experiment 4: Continuous time recurrent neural network under variating gravity

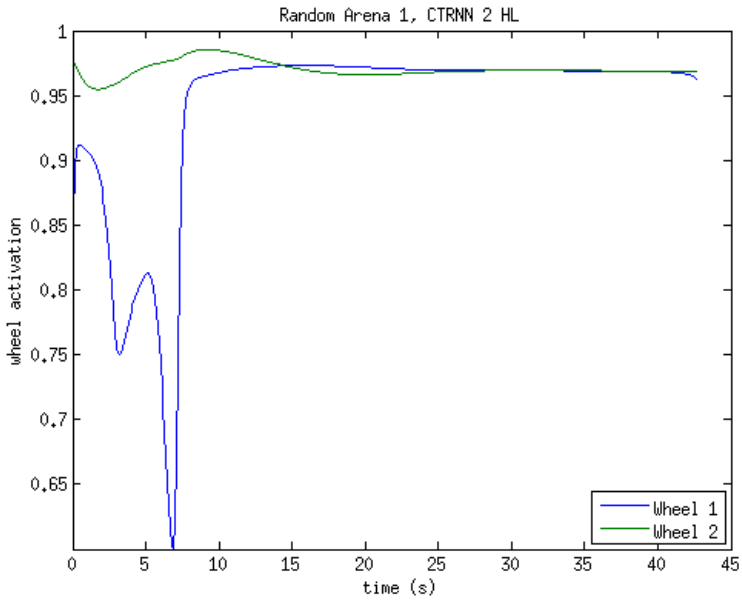
The positive results of the past experiments motivated us to start making the problem more complex. Until now we have been experimenting with a fix gravity in direction and in magnitude, not only but also the half of the arena that presented gravity was also fix during evolution, i.e. that the neurocontroller did not experience any variations in the environment during evolution. In this experiment we will start introducing some randomness and variation in the arena. We will start by randomly selecting which half of the arena will present gravity and also the direction of the force is no longer fix in north direction but changes randomly between north and south direction (i.e. the robot will drift upwards or downwards). Another aspect that complicates the experiment a bit more is the fact that the robot is no longer always initialized in the same position but is randomly initialized inside the following rectangle ($3 <= x <= 4$ & $-4 <= y <= 4$). The line where the two areas of the arena (gravity and no gravity) are separated is also randomly selected ($x = a,\ a \in [-2.5; 2.5]$). In order to expose the neurocontroller to a large number of cases during evolution we increased the number of samples for every generation from 1 in the past experiments to 20 in the future ones. To choose the best strategy to compute the fitness score we conducted several experiments with an average, best, and worst strategy. The best results were achieved with an average strategy, i.e. we compute an average of the 20 score values from the different samples. The fitness function remains the same as in equation 7. The resulting plots for three different cases and the activation of the output neurons (wheels) can be seen in figures 8 to 10. In this plots the purple arrow indicates the direction of the gravity and also the area in which it is placed signalizes on which half of the arena gravity is acting. The red arrow starting from the initial position of the robot indicates its initial orientation. In the wheel activation plots a value of 1 corresponds to full speed forward, a value of 0 to full speed backwards and a value of 0.5 full wheel stop (which does not mean that the robot is not moving, this would be the case if the robot is acting under gravity).

The CTRNN seems to be able to learn a general strategy during evolution that allows it during evaluation to be able to solve the problem and get to the light. What is a bit peculiar is the initial loop that can be observed for example in figure 8(a). The reasons for this behavior are not clear yet and given the nature and complexity of our controller it is not easy to find the underlying cause. This is perhaps one of the downsides of using artificial neural networks as a controller. A possible explanation could be the fact that the controller has to first recognize in which kind of environment it is being evaluated, since it has learnt during evolution that this one changes with every new initialization. This phenomenon is also reflect in the activation of the output neurons (figure 8(b)). In the first 15 seconds the activation experiences oscillations that account for the turning and at the same the compensation of the drift since there is no drifting of the robot towards the upper part of the arena (note that the drift is pointing in north direction).

This experiment shows that learning of a general strategy is possible, but we are still very far away of answering the question that motivated this work. Therefore the pursue the goal of making this setup more complex and design the next experiment.

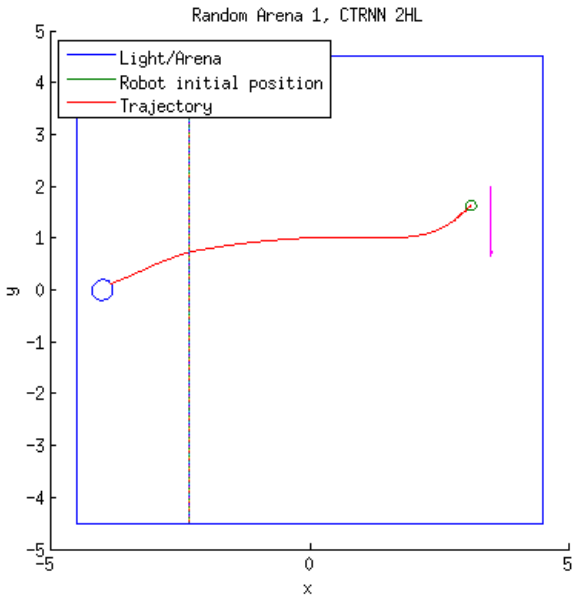


(a)

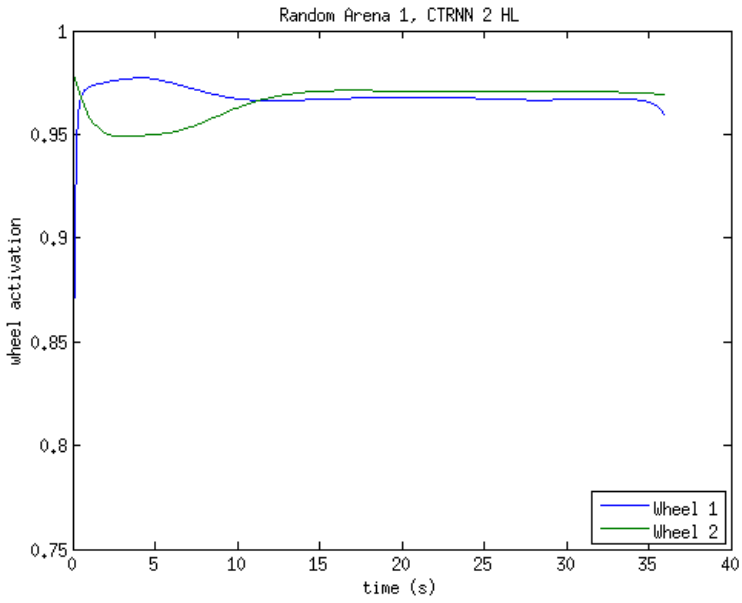


(b)

Figure 8: CTRNN with 2 hidden layers in a variating gravity environment: (a) Trajectory; and (b) Wheel activation.

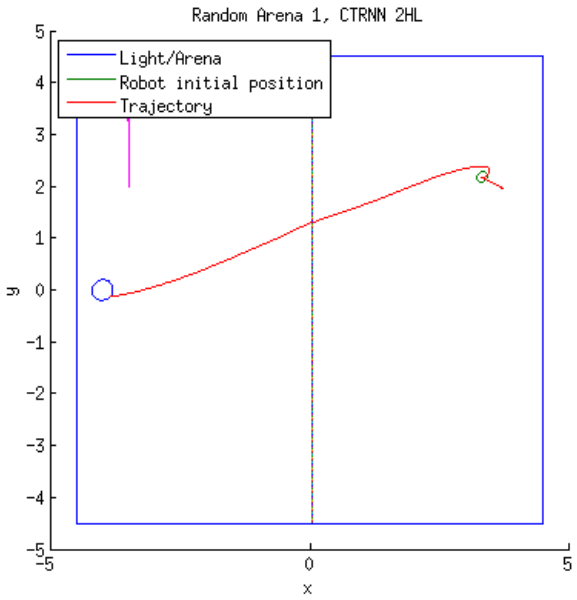


(a)

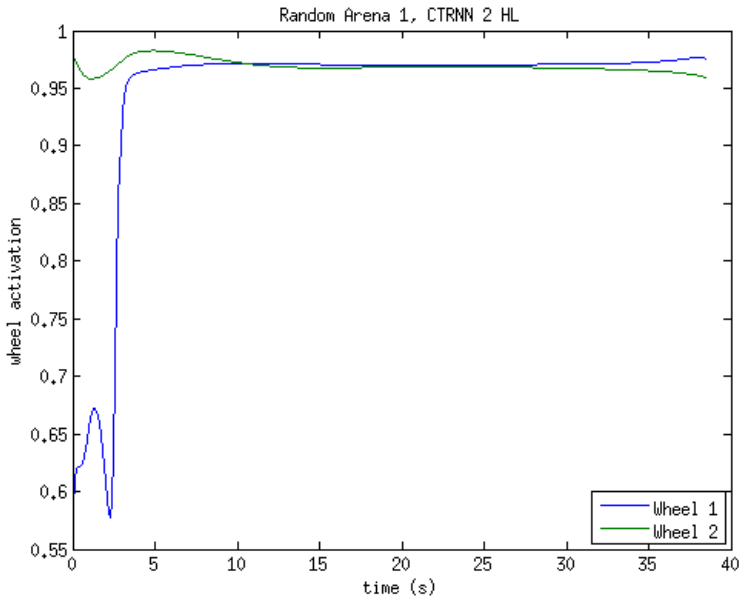


(b)

Figure 9: CTRNN with 2 hidden layers in a variating gravity environment: (a) Trajectory; and (b) Wheel activation.



(a)



(b)

Figure 10: CTRNN with 2 hidden layers in a variating gravity environment: (a) Trajectory; and (b) Wheel activation.

Experiment 5: Continuous time recurrent neural network under variating gravity and speed minimization

With the goal of making this setup more similar to a space application we decided to introduce a new factor into the behavior of the neurocontroller. The idea is to use the environment, in this case the gravity, to get to the light, and by doing so minimizing speed. It is obvious that in some cases this will not be possible, since depending on the relative position to the light, gravity might make the robot to drift away of the light source. In this case the neurocontroller should be able to recognize the situation and take the necessary countermeasures.

In order to be able to evolve the above described behavior we had to further develop the fitness function (equation 7) that we had been using for the past experiments. For this case the fitness function has to be complemented with one term that reflects the speed of the robot. Since we are working with a simulator and not with real robots that depend on a power source like a battery, the speed term in this case will be determined by the activation of the output neurons of the neurocontroller. The fitness functions takes now the form of equation 8:

$$f(t) = \frac{d_0 - d_\mathrm{s}}{d_0} \cdot (1 - \frac{W}{T_\mathrm{A}}) \tag{8}$$

where d_0 is the initial distance and d_s is the shortest distance to the light and W as:

$$W = \sum_{t=0}^{t_\mathrm{e}} (|W_1| + |W_2|) \tag{9}$$

with W_1 and W_2 being the activation values of the output neurons in every simulation step and T_A as:

$$T_\mathrm{A} = \frac{t}{t_\mathrm{step}} \tag{10}$$

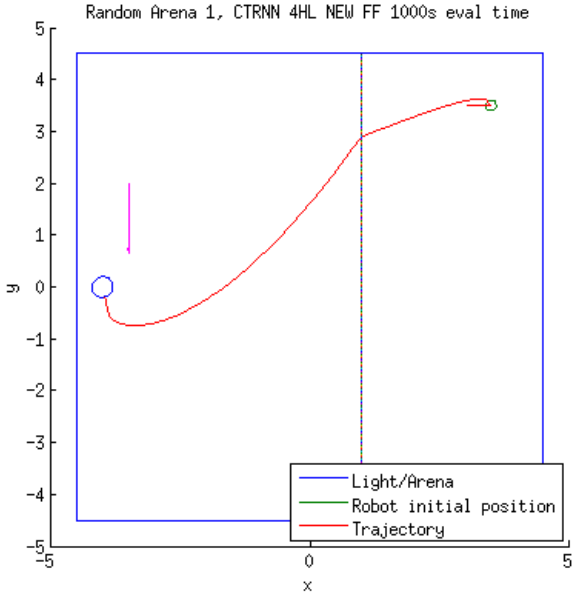
with t_e as the simulation time and t_step the time step length.

After making the proper modifications of the code and implementing the new fitness function we ran several evolutions and came to the conclusion that a modification of the GA parameters was needed. The reason is the following: since the controller is going to try and minimize speed, this means that the velocity with which the robot is going to displace itself through the arena is going to be much lower than with the old fitness function in equation 7. For this reason we had to increment the evaluation time to $t_\mathrm{eval} = 1000s$ in order to give evolution time enough to find a proper strategy. The structure of the network also suffer a minor modification, the number of hidden layers was increased from two to four. Previous experiments with the simpler of the networks showed to be insufficient to solve the problem. The environment and arena were kept the same as in the last experiment.

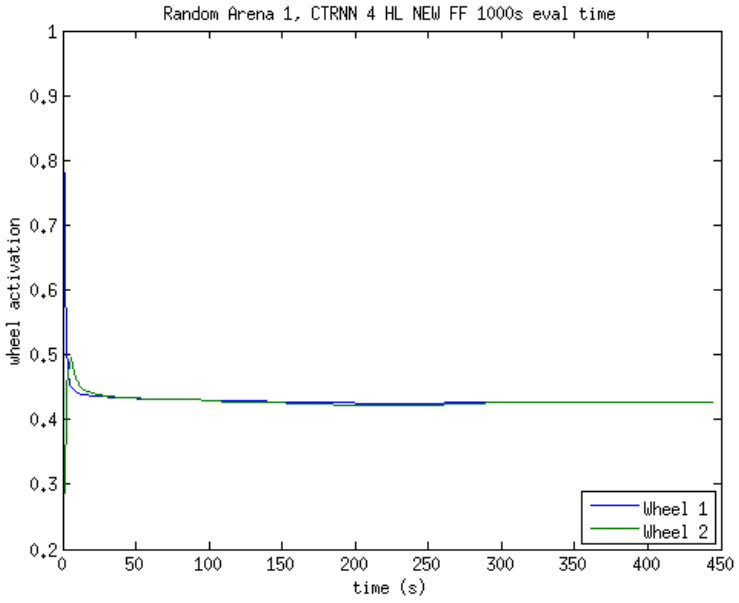
A large number of setups were considered for this particular experiment, we selected two cases to present in this report. The trajectory and wheel activation plots for a favorable situation (i.e. the relative position of the robot is so that taking advantage of the gravity takes it to the light) can be found in figure 11

and for an unfavorable situation (the gravity takes the robot away from the light) in figure 12.

We can conclude again that the evolved neurocontroller successfully developed a general strategy that solves the problem in both a favorable and unfavorable situations. A quick comparison of figures 11(b) and 12(b) with figure 10(b) from the past experiment shows that the neurocontroller is successfully minimizing speed too, since the activation of the output neurons is much lower in this case. Another point that is worth to be discussed is the similarity at a first glance of the wheel activation for the favorable and unfavorable case. To explain this we have to consider equation 10. In the calculation of the speed term it is not the activation of the output neurons that goes directly into this term but the sum of them over the whole simulation time. This is the reason why the difference between a favorable and unfavorable situation lies in the amount of time the robot needs to get to the light. The simulation of the favorable case ended almost $350s$ before the one for the unfavorable one, this accounting for a reduction of 50% in the duration of the trial. We can therefore conclude that evolution is developing a strategy that actually uses the environment for its advantage, producing results that show some light over the first question that motivated this experiments.

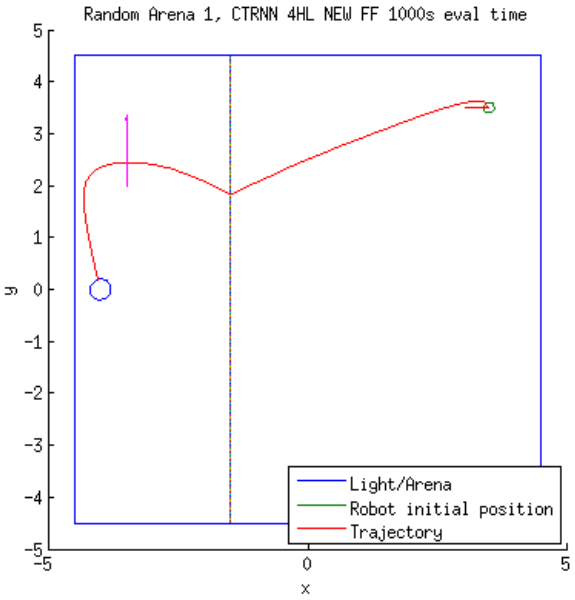


(a)

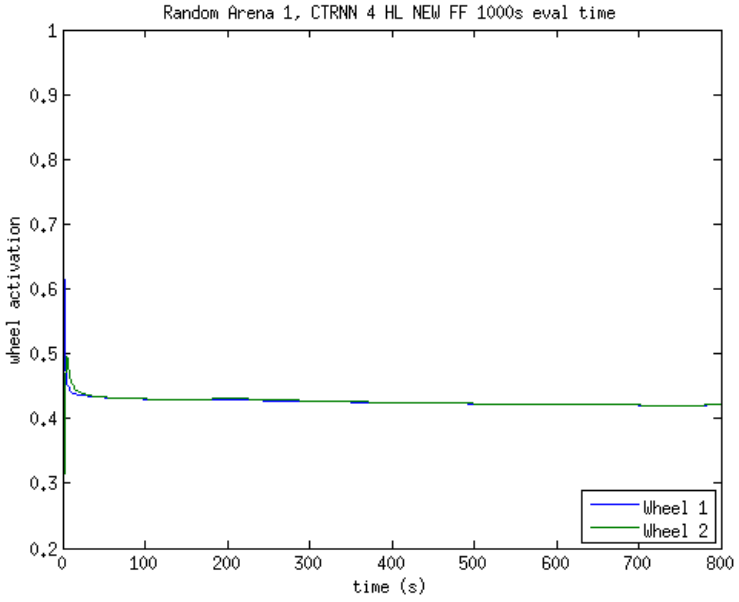


(b)

Figure 11: CTRNN with 4 hidden layers in a variating gravity environment (favorable situation) minimizing speed: (a) Trajectory; and (b) Wheel activation.



(a)



(b)

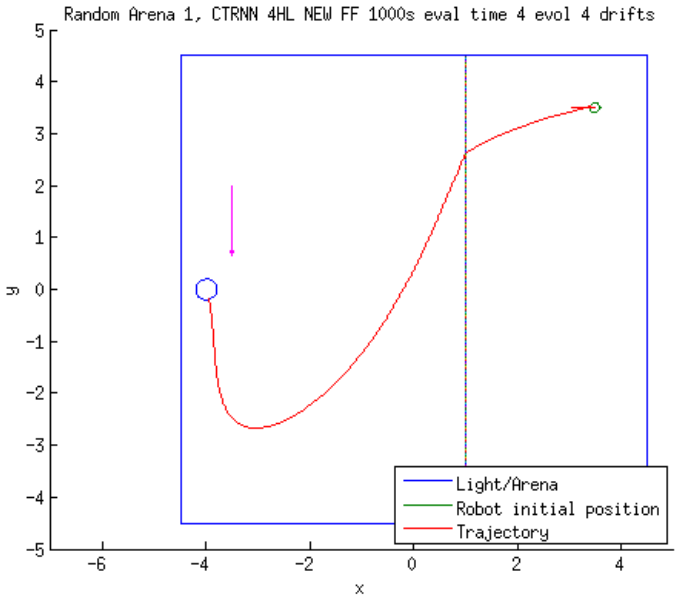
Figure 12: CTRNN with 4 hidden layers in a variating gravity environment (unfavorable situation) minimizing speed: (a) Trajectory; and (b) Wheel activation.

Experiment 6: Continuous time recurrent neural network under 4 directions of gravity and speed minimization

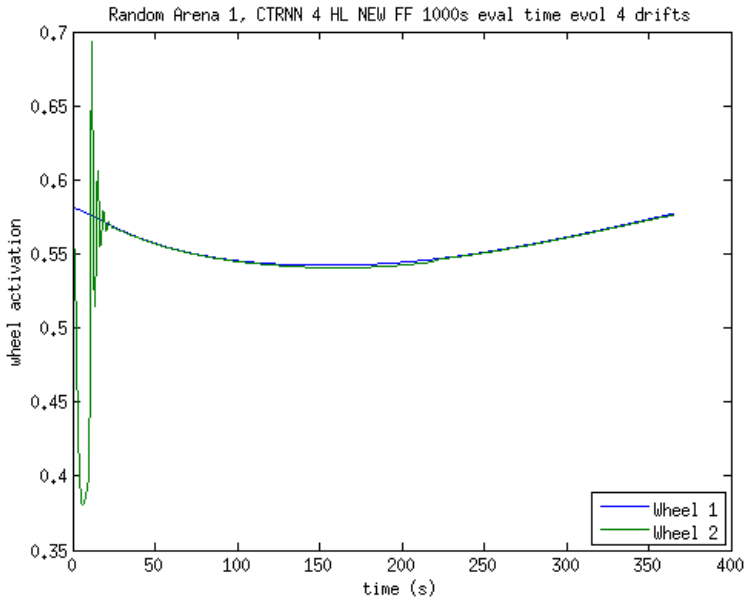
We are slowly getting closer to a task worth of being distributed under the BOINC infrastructure since the time needed for evolution keeps rising with the complexity of the problem. The last experiment took approximately 11 hours to evolved since very long evaluation times are needed. This rises the question in which extent the evaluation time plays a role in the quality of the evolved strategy. This among other issues could be addressed once the distributed computing environment is up and running in the ACT. Until this time arrives we want to continue gathering experience with this type of tasks and controllers and go one step further in the complexity of the experiments. It could be argued that we are not exposing our controllers with enough variability of the environment. For this reason we will complement experiment 5 with two extra directions of drift. This means that the controllers will be exposed randomly to a north, west, east or south gravity direction. We would expect in the case that the robot gets initialized at the same hight of the light and if gravity is pointing in west direction that the robot in the ideal case just drifts to the light without spending any fuel.

In figure 13 we can appreciate the performance of a CTRNN with 4 hidden layers. As we can observe in the trajectory plot in the first half of the arena the robot is confronted with a gravity free environment and produces a straight line trajectory to get closer to the light (also note the relative high value of the wheels activation in figure 13(b)). As soon as the barrier is crossed and the gravity environment starts the robot begins drifting towards the bottom of the arena, getting at the same time closer to the light and minimizing speed until $t = 200s$ when it starts sensing that its getting away from the light and starts using more speed in other to accomplish the task of getting to the light source. With this setup and since we do not want this report to be a collection of Matlab plots we jump directly to the case mentioned in the introduction of this experiment. After evolution the robot is placed in a full west gravity environment (i.e. the floor of the whole arena is moving towards the light), the resulting plots for trajectory and output neuron activation can be found in figures 14(a) and 14(b).

This result confirm the expected behavior. The neurocontroller recognizes a full favorable environment and starts minimizing speed from the beginning of the evaluation. This means that the evolved strategy allows the neurocontroller to use the environment for its full advantage. We want to remember the interested reader that an activation value of 0.5 means that no speed is being spent, i.e. the wheels are no being used. This means for this case that the controller is using less than 10% of the available power.

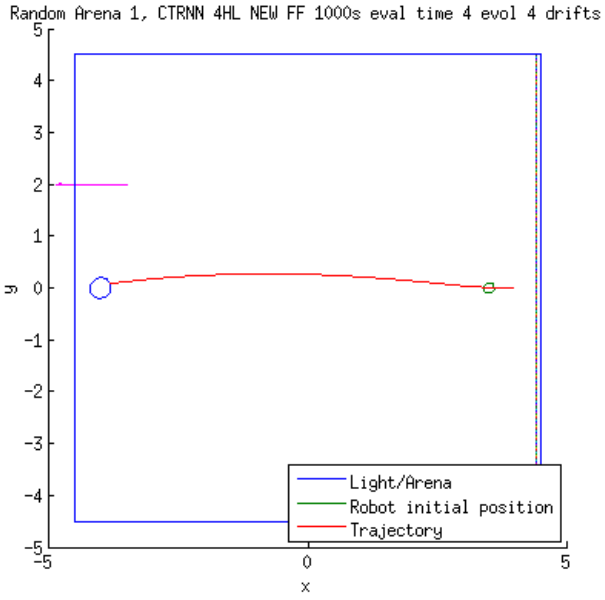


(a)

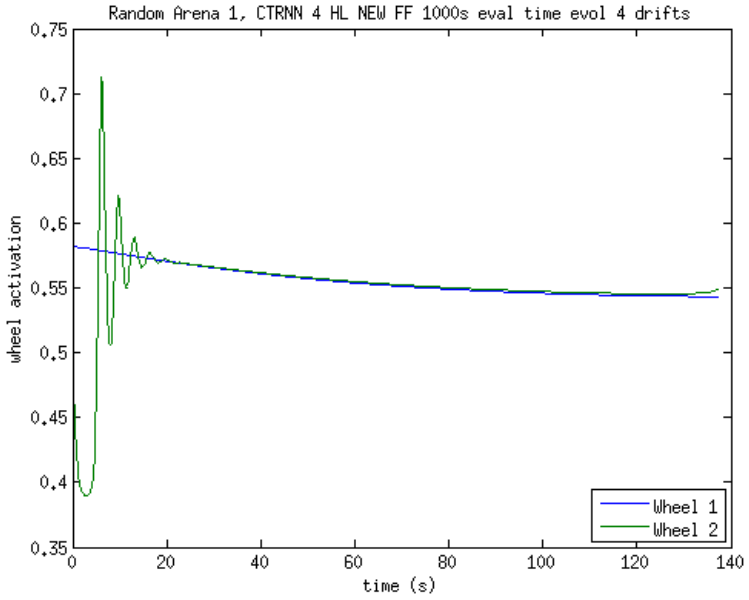


(b)

Figure 13: CTRNN with 4 hidden layers in a 4 directions of gravity environment (favorable situation) minimizing speed: (a) Trajectory; and (b) Wheel activation.



(a)



(b)

Figure 14: CTRNN with 4 hidden layers in a 4 directions of gravity environment (full west situation) minimizing speed: (a) Trajectory; and (b) Wheel activation.

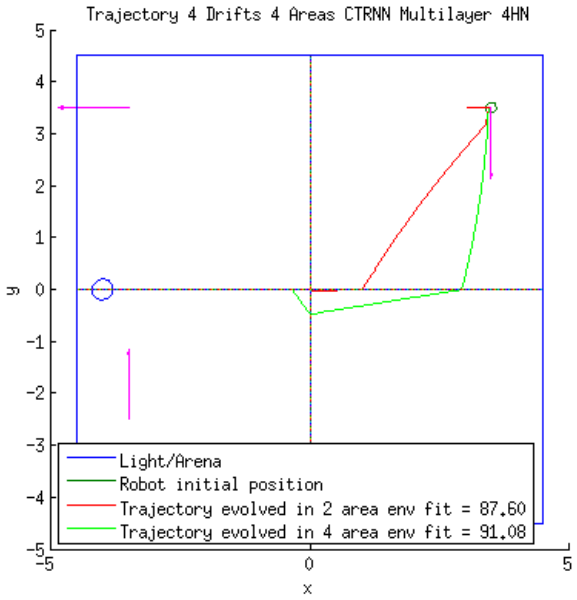
Experiment 7: Continuous time recurrent neural network under 4 directions of gravity in 4 different areas and speed minimization

In this experiment the arena is modified in a way that it has 4 different areas, in each of the areas the controllers can encounter a gravity environment in direction north, west, east or south, or a gravity free environment. We were interested in answering the question if an individual evolved in the previous setup could be able to solve the task in this more complex environment and if so to compare the trajectories and output neuron activation between this individual and an individual evolved in this 4 area environment. The results of this test for a favorable situation can be found in figure 15, 16 and 17, and for an unfavorable configuration of the gravity direction in figure 18 and 19.

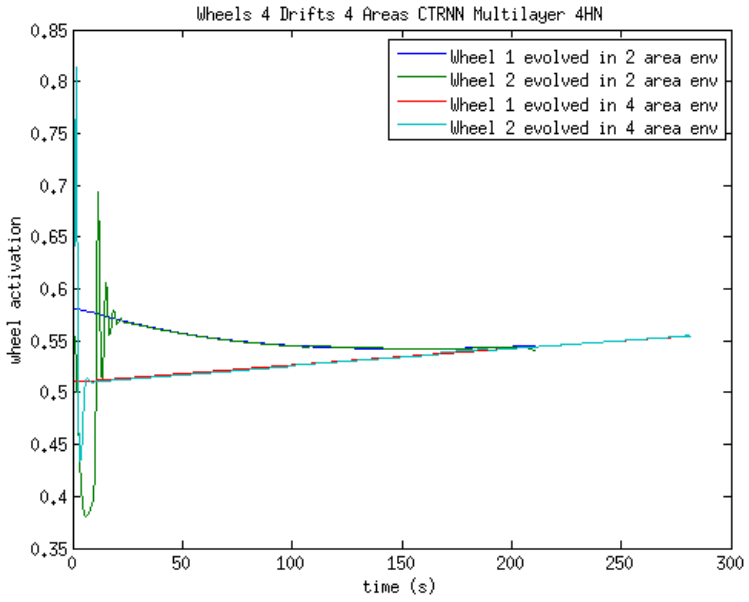
By comparing the above mentioned figures we can conclude that there is no significant difference between the behavior evolved in this configuration and the behavior evolved in the setup of the previous experiments. This means that the behavior learned in the 2 areas arena can be easily extrapolated to more complicated scenarios, e.g. the 4 areas arena. Another aspect that is also very interesting as a result of this experiment is the comparison of figures 16(b) and 18(b). In both figures we are seeing plots of the activation level of the output neurons, first for a favorable configuration of the arena and the initial position of the robot (i.e. the controller can use the drift for its advantage to get to the light) and second for an unfavorable configuration (i.e. the controller has to somehow fight drift to get to the light). These show that the controller is choosing, depending in which kind of environment it is initialized, a different strategy that allows it to get to the light and by doing so minimizing speed. If it can profit from the environment, it will do so by applying a lower activation level of the wheels (16(b)), if the drift is moving it away from the light it will have to spend more speed (i.e. a higher activation level of the wheels) to complete the task (18(b)). To support our conclusion several simulations were conducted with different initial positions and arena configurations. These results can be found in figures 15, 16, 17, 18 and 19.

Another effect worth commenting is the fact that the in some cases (e.g. figure 15(a)) the trajectory of the robot matches the line where two different drift areas are divided. During the execution of these experiments the cause of this effect was unknown. After changing the fitness function to evolve a fuel minimizing behavior (see equation 11) we came to the conclusion that this was due to the low speed of the robot. In fact the trajectory does not really match the division line, but is in reality a zig-zag line, i.e. the robot tries to overcome the drift, but since one of the goals is to minimize speed it can not acquire the necessary speed to overcome the drift.

In the next section we will report on the last configuration developed during this stage which is a 9 area environment first with 4 directions of gravity and second with a fully random gravity direction and magnitude.

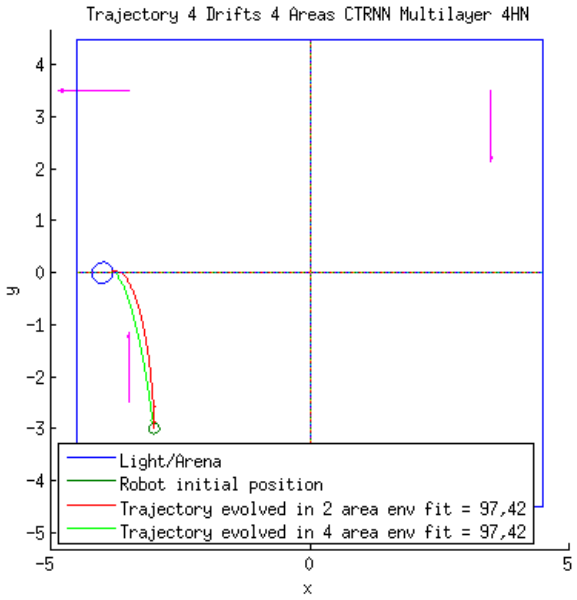


(a)

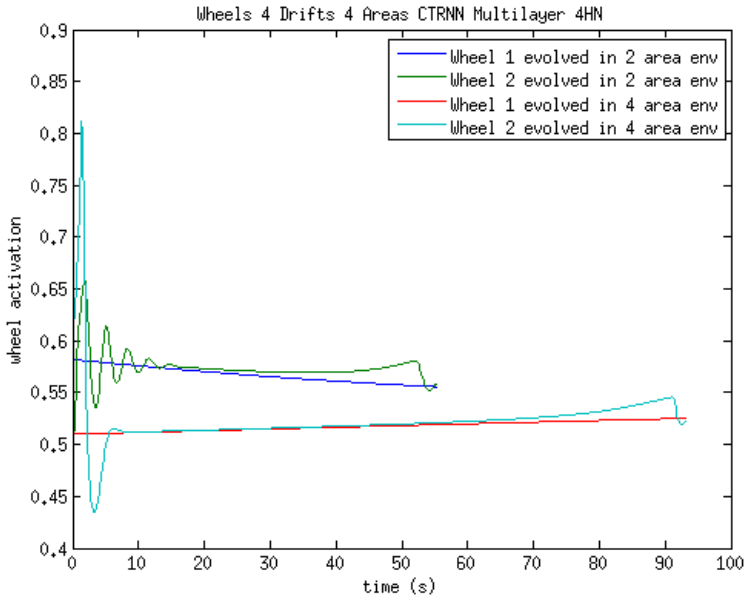


(b)

Figure 15: CTRNN with 4 hidden layers in a 4 directions of gravity environment (4 areas favorable setup) minimizing speed: (a) Trajectory; and (b) Wheel activation.

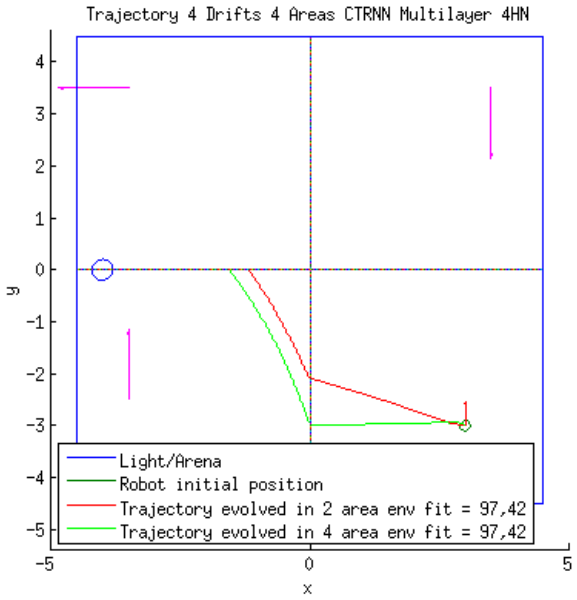


(a)

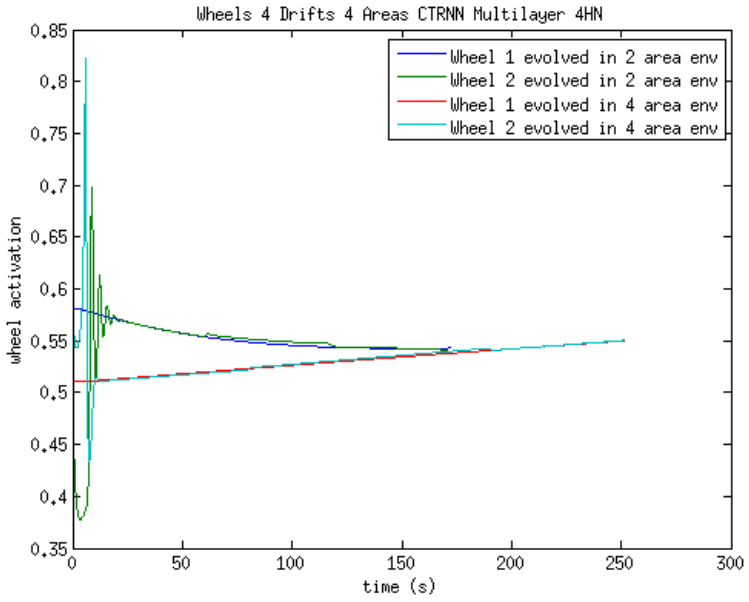


(b)

Figure 16: CTRNN with 4 hidden layers in a 4 directions of gravity environment (4 areas favorable setup) minimizing speed: (a) Trajectory; and (b) Wheel activation.

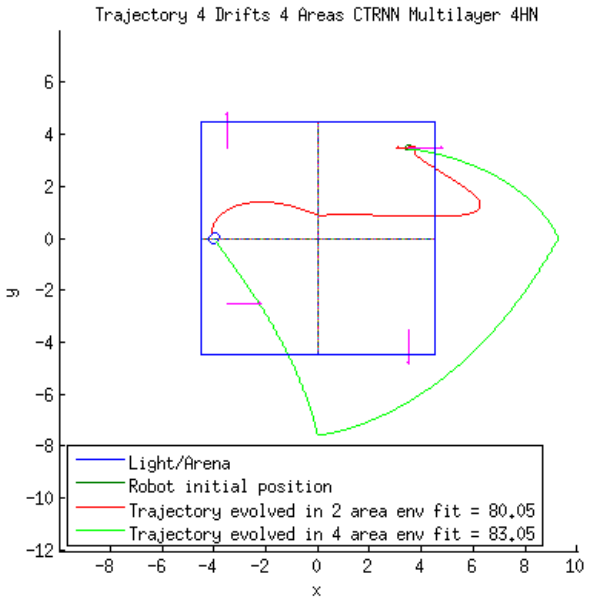


(a)

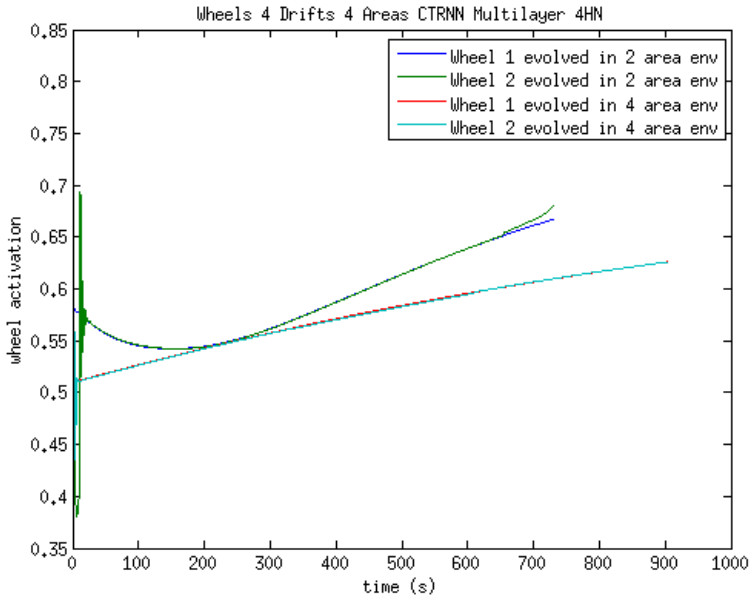


(b)

Figure 17: CTRNN with 4 hidden layers in a 4 directions of gravity environment (4 areas favorable setup) minimizing speed: (a) Trajectory; and (b) Wheel activation.

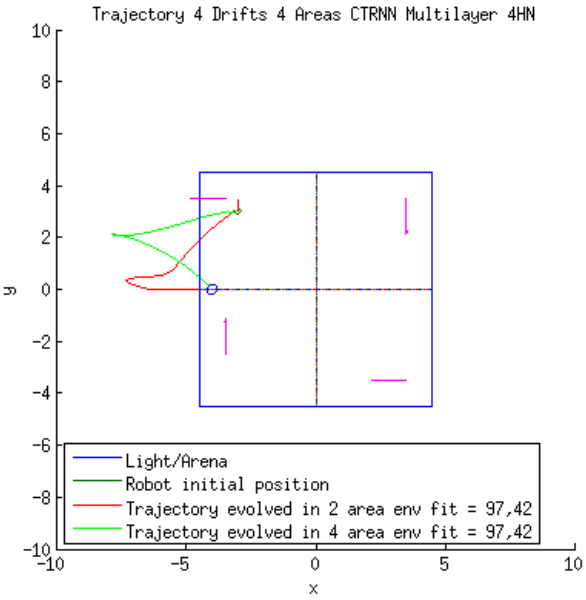


(a)

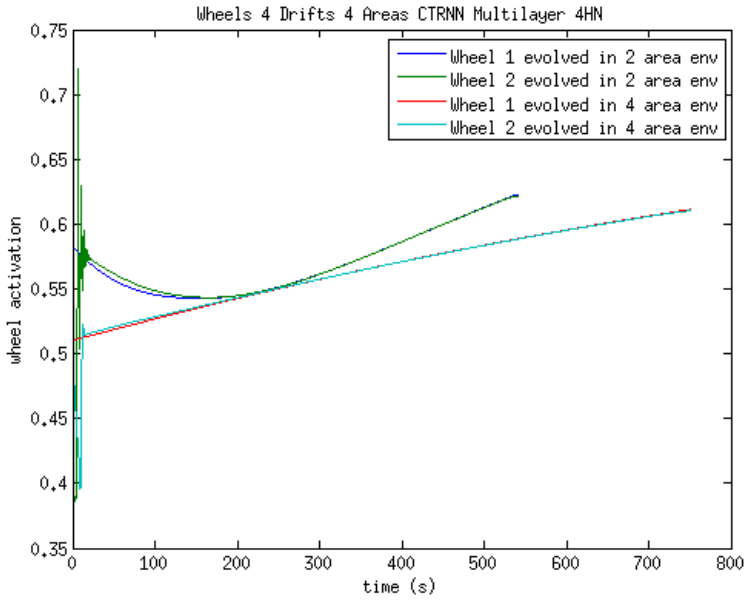


(b)

Figure 18: CTRNN with 4 hidden layers in a 4 directions of gravity environment (4 areas unfavorable setup) minimizing speed: (a) Trajectory; and (b) Wheel activation.



(a)



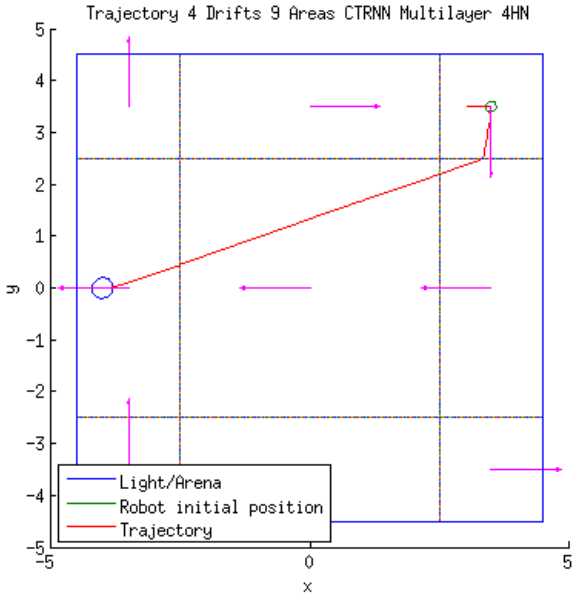
(b)

Figure 19: CTRNN with 4 hidden layers in a 4 directions of gravity environment (4 areas unfavorable setup) minimizing speed: (a) Trajectory; and (b) Wheel activation.

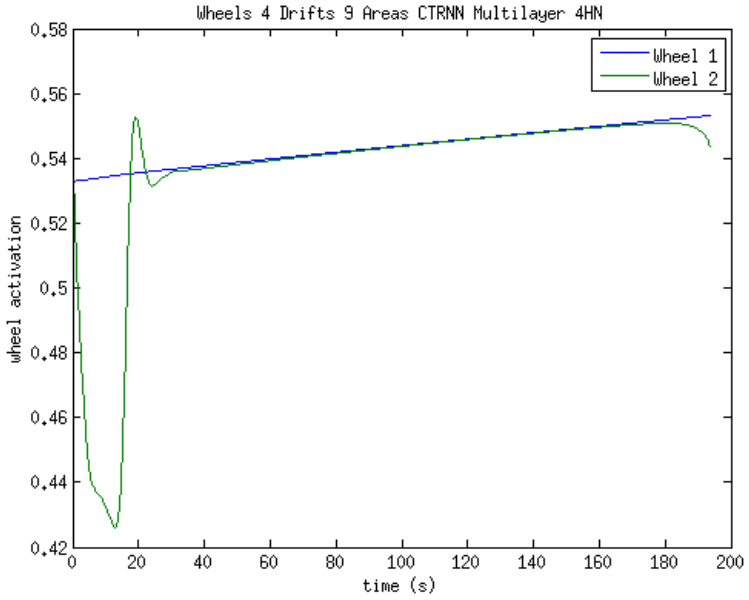
Experiment 9: Continuous time recurrent neural network under 4 gravity direction in 9 different areas and speed minimization

In this experiment we generalized the problem and go for a 9 areas arena with 5 different gravity conditions (north, west, south, east or none). The evolution is done as in the previous experiments, i.e. every individual is evaluated 20 times and the average fitness score is build. During the 20 trials the controller is initialized in different positions and the configuration of each square of the arena changes randomly. After evolution the configuration of the arena is fixed and the best evolved individual is evaluated. In figure 20 the evolved behavior in a favorable arena configuration (i.e. drift can get the robot to the light) can be observed.

The evolved behavior is very interesting if we consider that this individual has with a high probability never seen this arena configuration. The controller is accomplishing the task and at the same time minimizing speed. In the next section we will go one step further and confront evolution with fully random gravity direction and magnitude.



(a)



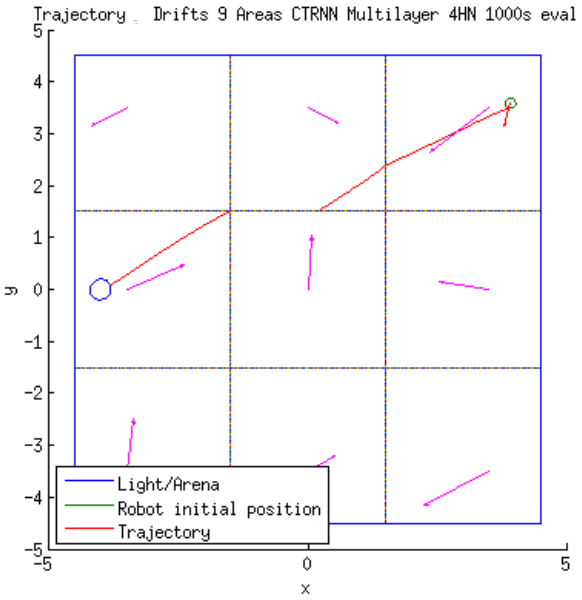
(b)

Figure 20: CTRNN with 4 hidden layers with 4 gravity directions minimizing speed (eval time 1000s): (a) Trajectory; and (b) Wheel activation.

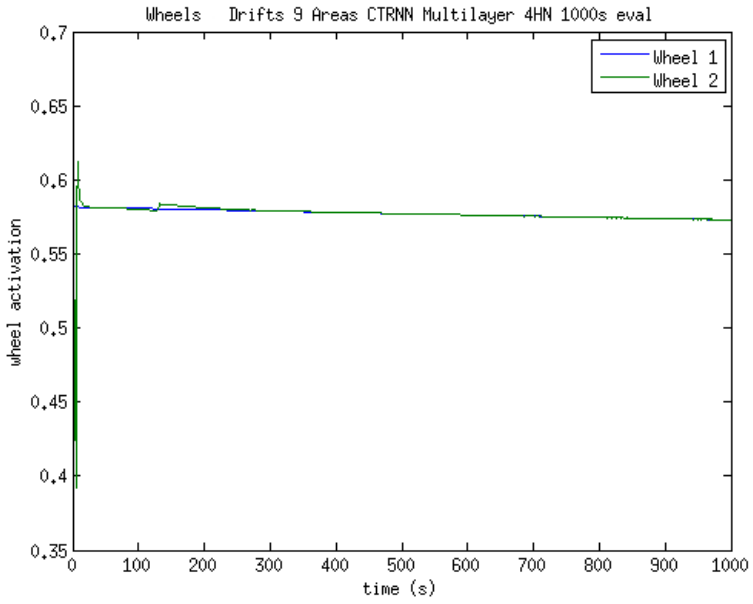
Experiment 10: Continuous time recurrent neural network under random gravity direction and magnitude in 9 different areas and speed minimization

In this section we will present the results of the last experiment with the fitness function from equation 8. The gravity present in the environment is fully random in direction and in magnitude. The direction varies during evolution between 0 and $2 \cdot \pi$ and the magnitude between 0 and $2 \cdot 0.03^{\frac{1}{2}}$. As we can observe in figure 21 the best individual that arrises from evolution after 100 generations is able to perform the phototaxis. It is worth mentioning once again that the arena changes with every evaluation of a chromosome, i.e. that the probability of exactly this individual being evaluated in an environment configuration that it has already experienced during evolution is very small. From this fact we can conclude that evolution is providing the neurocontroller with the ability to apply a general strategy that allows it to accomplish the given task. As we can see in figure 21(b) the evolved strategy also minimizes speed.

In the next section we will present a new fitness function that will allow us to evolve a fuel minimizing environment. We will also test the evolved strategy in a 4 areas 4 drifts environment, in a 9 areas 4 drifts environment and we will conclude our experiments with a 9 areas fully random drift in direction and magnitude environment.



(a)



(b)

Figure 21: CTRNN with 4 hidden layers with fully random gravity direction and magnitude minimizing speed (eval time 1000s): (a) Trajectory; and (b) Wheel activation.

Experiment 11: Continuous time recurrent neural network with a fuel minimizing fitness function

The idea behind the following experiments is to be able to evolve a behavior that not only performs phototaxis, but also minimizes fuel consumption. This has an enormous relevance for the successful planning and development of a space mission. In equation 11 we present a new fitness function that is a combination of two terms. The first term describing phototaxis has been already used in equations 7 and 8 where d_0 is again the initial distance to the light and d_s the shortest distance to the light. The second term describes the fuel minimization task, where $|W_1|+|W_2|$ is the sum of the absolute values of the activation values of the output neurons (this values are maped between -0.5 (full speed backward) and 0.5 (full speed forward)), and t_e is the simulation time. In equation 11 the phototaxis term can be a maximum of 100 (the robot gets to the light source) and a minimum of 0 (the robot does not move from its initial position). The fuel minimization term can be a maximum of 101 (the robot does not use any fuel) and a minimum of 1 (the robot uses its wheels all the time) since the sum of the absolute values of the wheel activations over a maximum of 1000 s of evaluation time with a sample time of 0.1 s is $10,000$. This gives a total minumum fitness score of 1 and a maximum of 201.

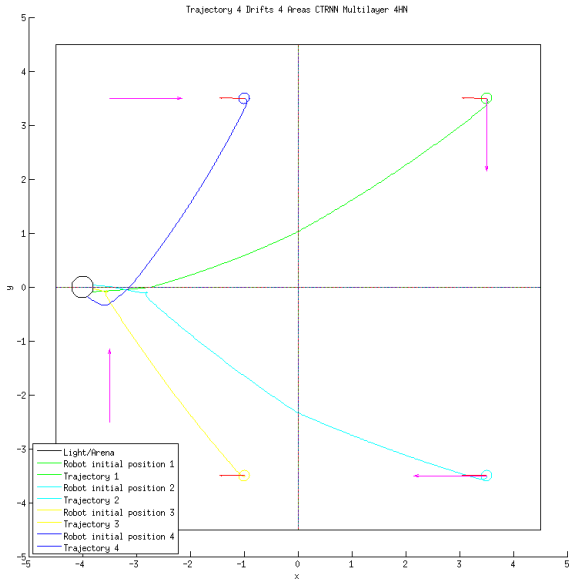
$$f(t) = 100 \cdot \frac{d_0 - d_s}{d_0} + \frac{10100}{\sum_{t=0}^{t_e} (|W_1| + |W_2|) + 100} \tag{11}$$

The first experiment that we performed with equation 11 was with a 4 areas 4 drifts arena configuration analog to ones performed in experiment 7. The resulting plots for trajectory and wheel activation can be seen in figure 22. The experiment was performed for 4 different initial position of the robot. The trajectories in figure 22(a) and the wheel activations in 22(b) are color coded so that the color of the trajectory equates the color of the wheel activation for a specific initial position. It is known form the optimal control theory that the best controller to solve this task would be a bang-bang controller. A behavior that could be interpreted as similar to this strategy can be seen in figure 22(b). This means that evolution is successfully accomplishing phototaxis and at the same time applying an optimal control strategy to solve the task.

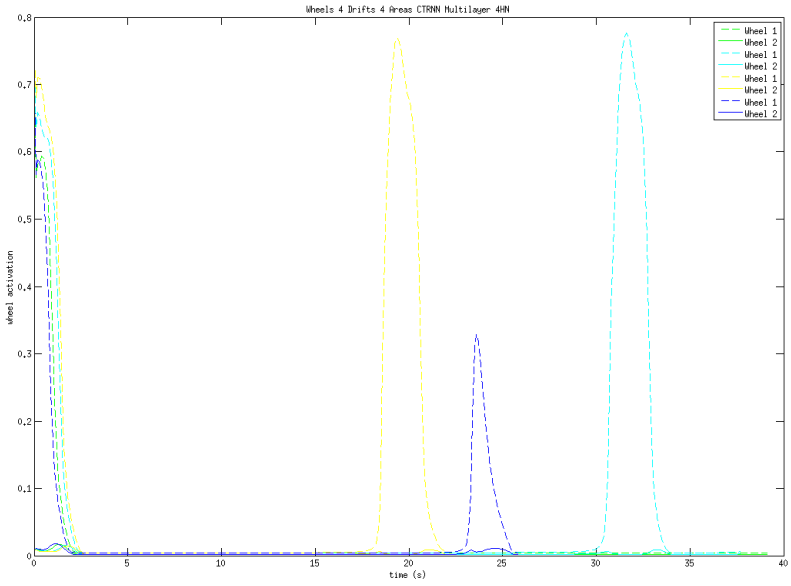
Following the same structure as in the last experiments, the next arena configuration that we tested is a 9 areas 4 drifts directions environment. The results for this experiment can be found in figure 23. As for the previous setup we evaluated the evolved individual from 4 different initial positions. We would expect that every time that the robot enters a new area with different gravity conditions as the previous one it corrects its relative orientation towards the light. This phenomenon is reflected in a bend in the trajectory (figure 23(a)) and in a correction of the wheel activation (figure 23(b) and more clearly in figure 25(a)).

To finalize our work with this robot model we present the results of the experiments performed with a 9 areas full random drift in direction and magnitude. The direction varies during evolution between 0 and 2π and the magnitude between 0 and $2 \cdot \sqrt{0.03}$. The plots for trajectory and wheel activation can be found in figure 24. For a better visualization of transition from one gravity configuration to the next one a zoomed plot of figure 24(b) can be found in 25(b)

All these experiments have been performed with a robots model with wheels. Another interesting space application would be to try to evolve adaptive behavior with a thrusters model. An interesting application of such a model could be a docking task with a space station. In our case the robot would be the space craft and the light the docking station. Naturally this problem is a step more complicated than the one that we have been trying to solve since for a docking problem it is also important the speed and the orientation that the spacecraft has when it arrives to the space station. This means that we will have to modify the fitness function and of course the physics of our simulator. In the next section we will summarize the work performed in this area.

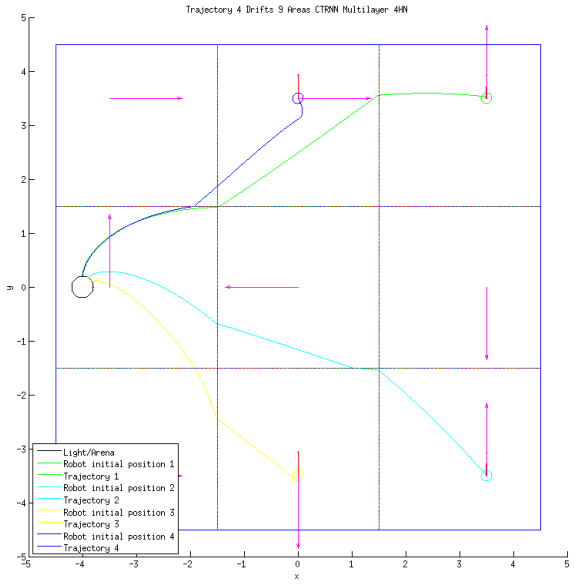


(a)

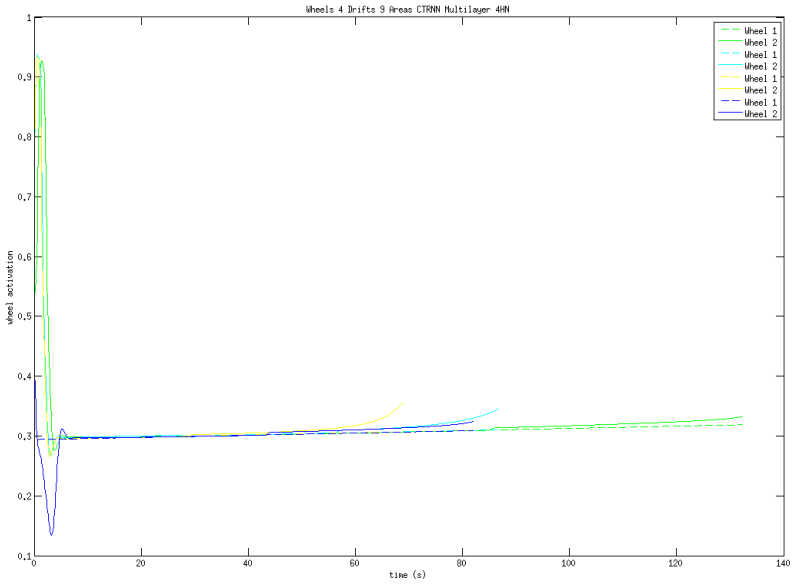


(b)

Figure 22: CTRNN with 4 hidden layers in a 4 areas 4 drifts environment and fuel consumption minimization(eval time 1000s): (a) Trajectory; and (b) Wheel activation.

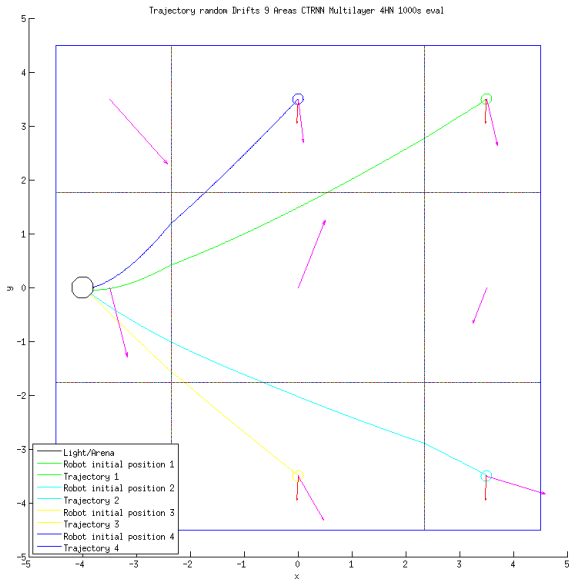


(a)

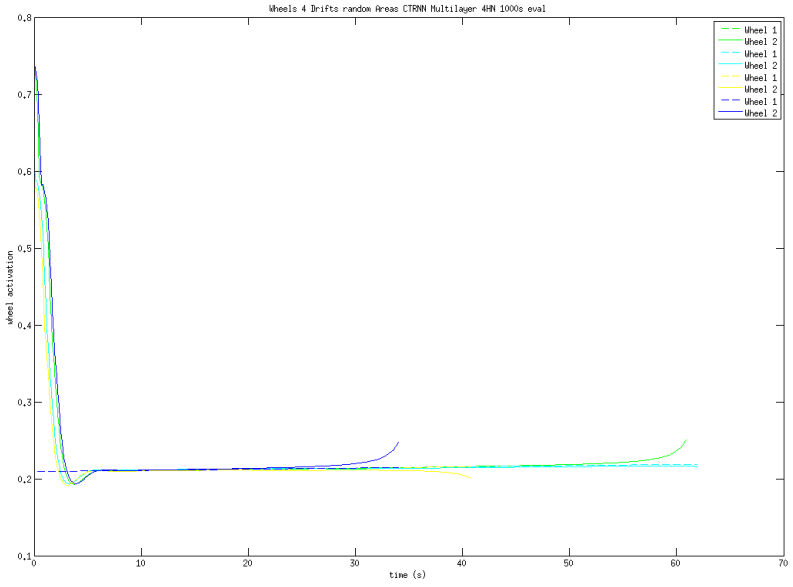


(b)

Figure 23: CTRNN with 4 hidden layers in a 9 areas 4 drifts environment and fuel consumption minimization (eval time 1000s): (a) Trajectory; and (b) Wheel activation.

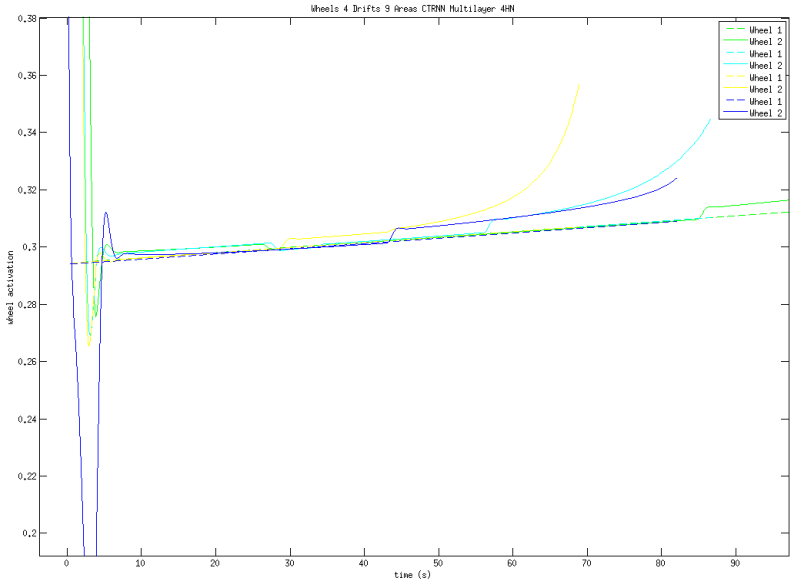


(a)

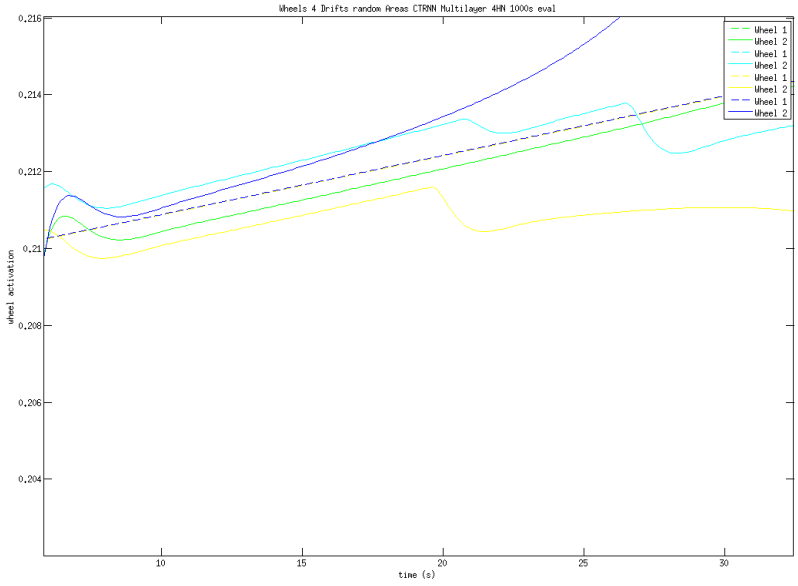


(b)

Figure 24: CTRNN with 4 hidden layers in a 9 areas fully random drift in direction and magnitude environment and fuel consumption minimization (eval time 1000s): (a) Trajectory; and (b) Wheel activation.



(a)



(b)

Figure 25: Zoom of figure 23(b) and figure 23(b): (a) Wheel activation for a 9 areas 4 drifts environment; and (b) Wheel activation for a 9 areas random drifts environment.

Thruster model

The thruster model implemented in this work is a dynamical model with tidal gravity, the theoretical basis and full description of this model can be found in [9].

$$x_{i+1} = x_i + \dot{x}_i \cdot dt \tag{12}$$

$$y_{i+1} = y_i + \dot{y}_i \cdot dt \tag{13}$$

$$\theta_{i+1} = \theta_i + \dot{\theta}_i \cdot dt \tag{14}$$

$$\dot{x}_{i+1} = 2 \cdot \eta \cdot \dot{y}_i + 3 \cdot \eta^2 \cdot x_i + (u_L + u_R) \cdot cos(\theta_i) \tag{15}$$

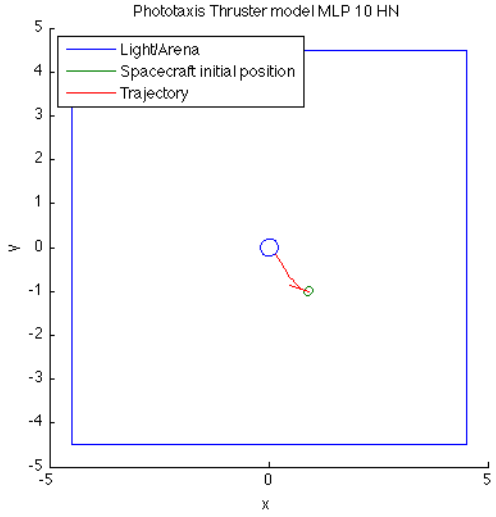
$$\dot{y}_{i+1} = -2 \cdot \eta \cdot \dot{x}_i + (u_L + u_R) \cdot sin(\theta_i) \tag{16}$$

$$\dot{\theta}_{i+1} = \frac{L}{I} \cdot (u_L - u_R) \tag{17}$$

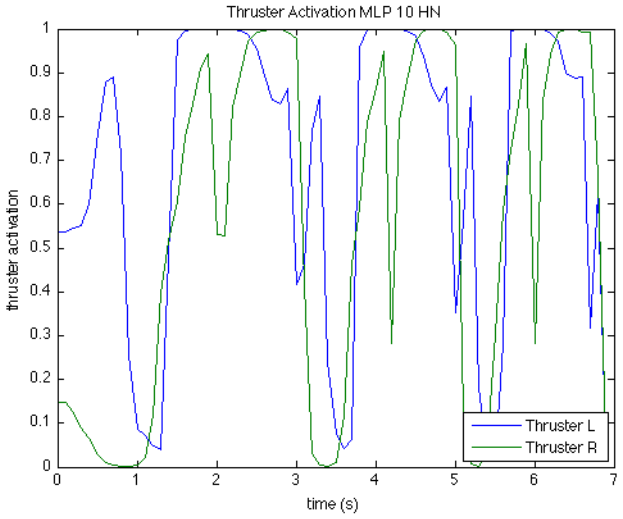
Here x_i and y_i represent the position in a 2-dimensional arena of the spacecraft, θ_i the orientation, $\dot{x}_i$ and $\dot{y}_i$ the velocity in x- and y-direction respectively, $\dot{\theta}_i$ the angular velocity and dt the sample time. The constant η describes the influence of the tidal gravity, u_L and u_R are the control variables (in this case left and right thrusters).

For time reasons we could only perform simple phototaxis experiments with this model. The controller selected for this experiments was a Multi Layer Perceptron with 10 hidden neurons. The first step was to test the model without tidal gravity, the evolution was performed over 100 generations, with a population of 100 individuals with and evaluation time of $1000s$ and 1 trial for every chromosome. As a fitness function equation 7 was used. The trajectory and thruster activation plots can be found in figure 26. As we can clearly observe the evolved behavior solves without any major difficulties the phototaxis problem.

The next experiments were performed with a random initialization of each individual first in a circle centered around the light ((0,0)) with radius varying between 1 and 2 and second with radius varying between 2 and 3. The average of the obtained fitness values over 20 trials was performed. In the four results presented in figure 27 to 30 the evolved behavior performs the phototaxis correctly or gets very near the light. A reason for this could be that the evolution was performed only for 100 generations. We believe that further experiments for this particular setup have to be performed (e.g. evolution for more than 100 generations, more complex control structures, i.e. increasing the number of hidden neurons).

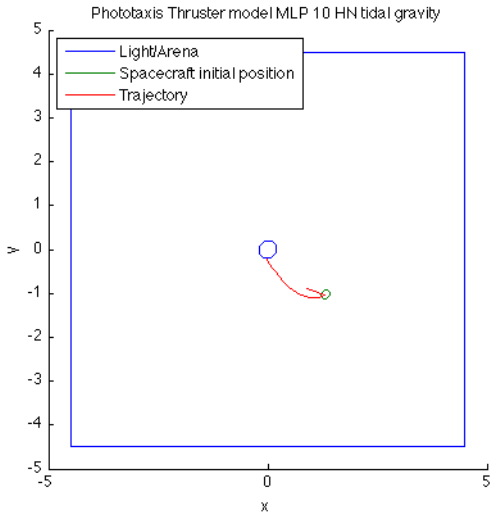


(a)

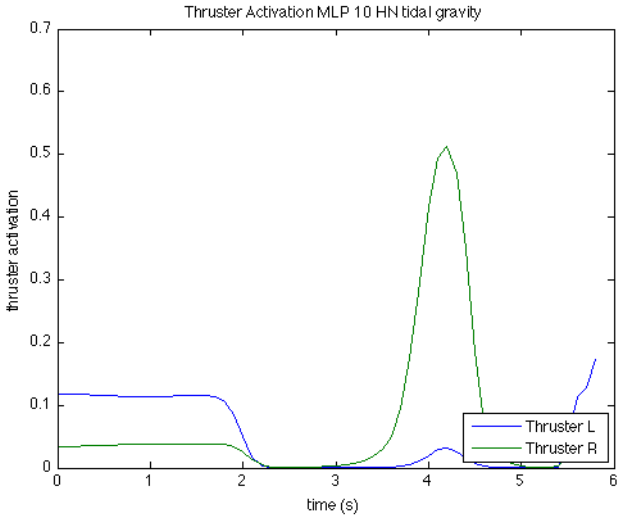


(b)

Figure 26: Phototaxis with a Multi Layer Perceptron with 10 hidden neurons, 1 evaluation trial: (a) Trajectory; and (b) Thruster activation.

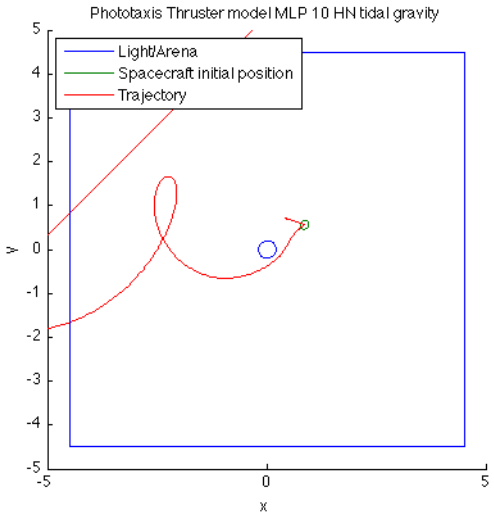


(a)

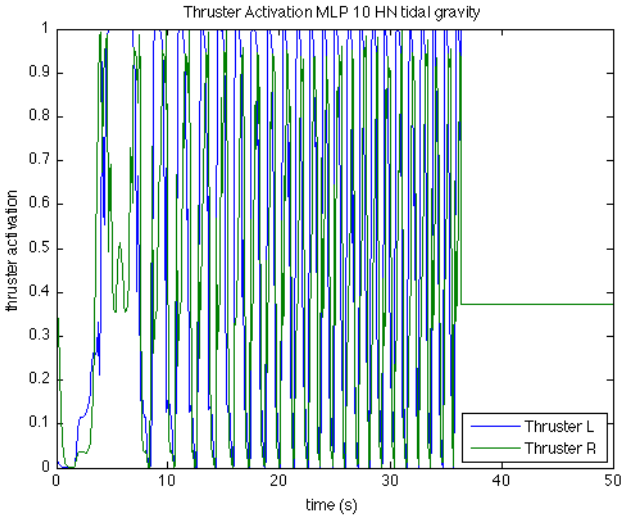


(b)

Figure 27: Phototaxis with a Multi Layer Perceptron with 10 hidden neurons, 20 evaluation trials (initial distance between 1 and 2) and tidal gravity ($\eta = 0.08$): (a) Trajectory; and (b) Thruster activation.

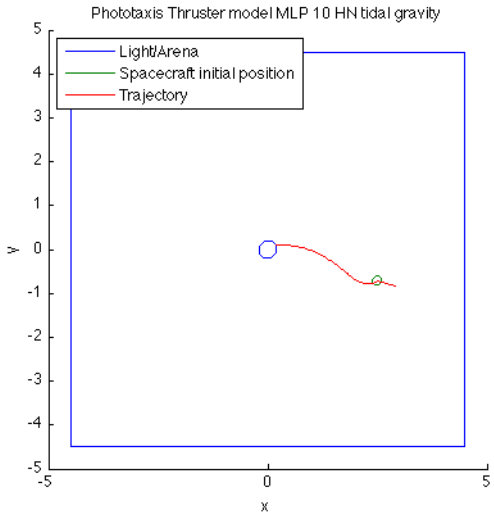


(a)

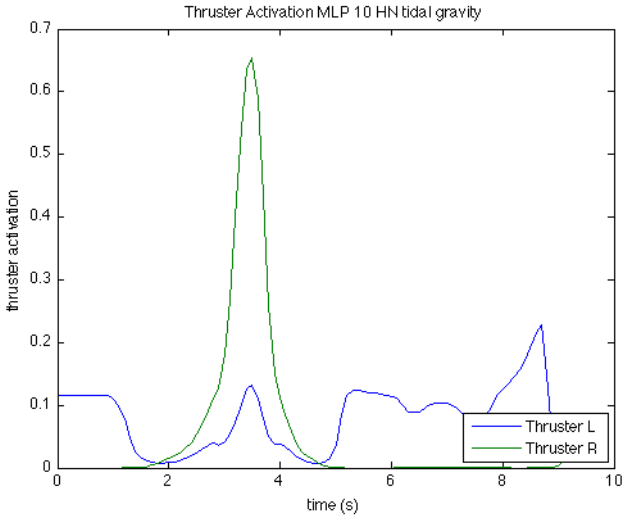


(b)

Figure 28: Phototaxis with a Multi Layer Perceptron with 10 hidden neurons, 20 evaluation trials (initial distance between 1 and 2) and tidal gravity ($\eta = 0.08$): (a) Trajectory; and (b) Thruster activation.

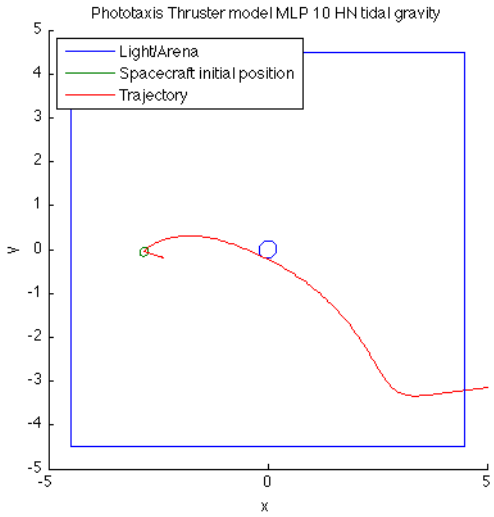


(a)

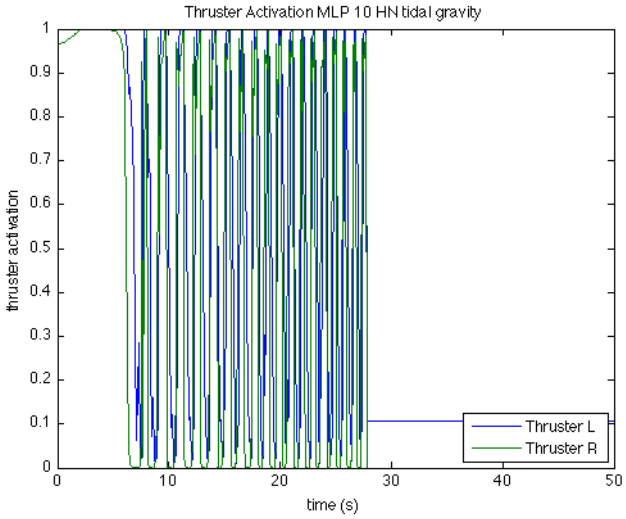


(b)

Figure 29: Phototaxis with a Multi Layer Perceptron with 10 hidden neurons, 20 evaluation trials (initial distance between 2 and 3) and tidal gravity ($\eta = 0.08$): (a) Trajectory; and (b) Thruster activation.



(a)



(b)

Figure 30: Phototaxis with a Multi Layer Perceptron with 10 hidden neurons, 20 evaluation trials (initial distance between 2 and 3) and tidal gravity ($\eta = 0.08$): (a) Trajectory; and (b) Thruster activation.

Conclusion

During the course of this stage we tried to answer some of the questions regarding the evolution of autonomous behavior. We tested different controllers in different configurations in the hope of having gained further knowledge in this area. In particular we were interested in the ability of a neurocontroller to develop a general strategy that allows it to find a solution to a given task in a varying gravity environment. A thruster model under the influence of tidal gravity was implemented to allow the test of more realistic space applications.

We believe that future research could take place in the development of fitness functions (e.g. for docking problems). The final velocity and orientation of the spacecraft will play an important role and has to be taken into consideration for the successful evolution of this behavior. The role that every evolution parameter plays in the evolved behavior has to be further analyzed. An interesting point would be to know how the number of generations influences the performance of the robot/spacecraft since we have seen e.g. that the evolved behavior in the case of the thruster model is not yet optimal in the sense that the evolved neurocontroller not always gets to the light source. During the course of our experiments we introduced randomness in the evolution by varying the configuration of the arena (i.e. gravity direction and magnitude) every time that a performance was evaluated, i.e. that every single individual after every single trial was evaluated under different conditions. The fact that this strategy might be unfair for the different individuals in a particular generation has been brought to our attention. In order to ensure a more fair comparison, an alternative to this strategy would be to change the conditions of the arena after every generation, i.e. every individual of a generation will be evaluated according to the same environmental conditions. The next aspect that requires further development is the generalization of the arenas. At the moment the different experiments can be performed but a new compilation of TwoDee has to be performed every time that the number of areas of which an arena is composed wants to be changed. This is not compatible with the distributed computing environment that is currently under development at the ACT. Ideally the user should be able to change the configuration of the arena directly from the command line and no new compilation of the source code should be necessary.

I would like at this point to thank all the members of the ACT for their support during the whole course of the stage, for an incredible first day joke that I will probably not be able to forget and also for making me feel welcomed from the first moment on.

Appendix A

```
from PyGMO import *
from numpy import *
from math import *
import time
import os
import string
from scipy import stats

class test_app:
        """
         Arguments: 1. algorithm: 'de(500, .8, .8, 2)'
        2. problem: 'schwefel(205)'
        3. #islands: '5'
     4. #individuals on each island: '20'
     5. #times the evolve command gets called: '200'
     6. #trials: '40'
     7. migration: 'migration(migration_scheme(0, 1, \
     topology.one_way_ring()),migration_policy(1, \
     choose_best_selection_policy(1), \
     random_replacement_policy(1)))'
     8. Mean Fitness: '0.01'
     9. Stdv Fitness: '0.1'
    10. Name: 'Test_Number_1'
         All arguments besides Name are by default empty strings, \
         Name is by default yyyymmddhhmmss, so that you can load \
         all the necessary information from a text file using the \
         read_test function.
         Output is saved to a file called: test_info_NAME.txt
         To create a test class: test_name = test_app()
         To run a test with the above parameters: \
         test_name.run_test()
         To read and load a testfile: \
         test_name.read_test('filename.txt')
         To save the results of the test: \
         test_name.save_test('filename.txt') default filename = \
         yyyymmddhhmmss
         To run a Welch's Test: test_name.welch_test('filename2.txt', \
         'filename_2.txt')
         """
         def __init__(self, algo = '', prob = '', n_isl = '', n_ind = '',\
          n_f = '', n_tri = '', migr = '', mean = '', std = '', \
          name = time.strftime("%Y%m%d%H%M%S", time.gmtime())):
 #Parsing the strings for the algorithm and problem
 self.algo_name = string.lower(algo)
 self.prob_name = string.lower(prob)
 self.n_isl = n_isl
```

```
self.n_ind = n_ind
self.n_f = n_f
self.n_tri = n_tri
self.migr = migr
self.mean_fit = mean
self.std_fit = std
self.name = name

##############################################################################
        def __repr__(self):
report_repr = 'Algorithm = ' + self.algo_name + '\nProblem = ' \
+ self.prob_name + '\nNumber of Islands = ' + str(self.n_isl) + \
'\nNumber of Individuals = ' + str(self.n_ind) + \
'\n#evolve gets called = ' + str(self.n_f) + '\nTrials = ' + \
str(self.n_tri) + '\nMigration = ' + self.migr
return report_repr

##############################################################################

        def run_test(self, verbose = False):
#Check if the file where the output is going to be saved to \
already exists, if it does you have to rename the class
        exec('self.algo = algorithm.' + self.algo_name)
        exec('self.prob = problem.' + self.prob_name)
        if self.migr != '':
 exec('self.migration = ' + self.migr)
        best_fit = []
        for j in range (0, int(self.n_tri)):
 if verbose:
        print 'Trial number: %d' % j
 i = 0
 if self.migr == "":
        self.archi = archipelago(self.prob, self.algo, int(self.n_isl),\
         int(self.n_ind))
 else:
        self.archi = archipelago(self.prob, self.algo, int(self.n_isl),\
         int(self.n_ind), self.migration)
 while i <= int(self.n_f):
        self.archi.evolve()
        self.archi.join()
        i = i +1
 best_fit.append(self.archi.best().fitness)
        self.mean_fit = 0
        self.stdv_fit = 0
        #Calculation of mean and stdv
        self.mean_fit = mean(best_fit)
        self.stdv_fit = std(best_fit)

##############################################################################
```

```
        def save_test(self, save_name = 'test_info_' + \
        time.strftime("%Y%m%d%H%M%S", time.gmtime()) + '.txt'):
#Checking if the output file already exists
if os.path.exists('test_info_' + self.name + '.txt') == 0:
 report = ['Date=', time.strftime("%Y%m%d%H%M%S", time.gmtime()), \
 '\nName=', self.name, '\nAlgorithm=', self.algo_name, '\nProblem=',\
 self.prob_name, '\nNumber of Islands=', str(self.n_isl), \
 '\nNumber of Individuals=', str(self.n_ind), '\n#evolve gets called=', \
 str(self.n_f), '\nTrials=', str(self.n_tri), '\nMigration=', self.migr, \
 '\nMean=', str(self.mean_fit), '\nStdv=', str(self.stdv_fit)]
 f = open(save_name, 'w')
        f.writelines(report)
        f.close()
else:
        print("The file already exists, please choose another name!")
```

```
##########################################################################
```

```
        def read_test(self, filename):
t = []
#Reading the file into a list
f = open(filename,'r')
for line in f.readlines():
        t.append(string.split(line, '='))
f.close()
#making a dictionary out of the list
d = {}
for j in range (0, size(t)/2):
        d[t[j][0]] = t[j][1]
s = []
#filtering the brakeline command out of the strings
for value in d.values():
        s.append(string.replace(value, '\n', ''))
i = 0
#copying the corrected values into the dictionary
for key in d.keys():
        d[key] = s[i]
        i = i + 1
#Creation of the PaGMO Objects from the file
self.algo_name = d.get('Algorithm')
self.prob_name = d.get('Problem')
self.n_isl = d.get('Number of Islands')
self.n_ind = d.get('Number of Individuals')
self.n_f = d.get('#evolve gets called')
self.n_tri = d.get('Trials')
self.migr = d.get('Migration')
self.mean_fit = d.get('Mean')
self.std_fit = d.get('Stdv')
self.name = d.get('Name')
```

```
##############################################################################

        def welch_test(self, filename, filename1):
#Two classes for the two files
cl = test_app()
cl1 = test_app()
#Loading the test info from the text-file to the abvobe created classes
cl.read_test(filename)
cl1.read_test(filename1)
#variables for the welch test (mean, stdv and samples for every variable)
X_mean = double(cl.mean_fit)
X_stdev = double(cl.std_fit)
X_count = double(cl.n_tri)
Y_mean = double(cl1.mean_fit)
Y_stdev = double(cl1.std_fit)
Y_count = double(cl1.n_tri)
#Welch's Test
t_value = abs(X_mean - Y_mean) / sqrt((X_stdev * X_stdev) / (X_count) +\
 (Y_stdev * Y_stdev) / (Y_count))
degrees_of_freedom = floor(pow((X_stdev * X_stdev) / (X_count) + \
(Y_stdev * Y_stdev) / (Y_count), 2) / (((pow(X_stdev, 4)) / \
(X_count * X_count * (X_count - 1))) + ((pow(Y_stdev, 4)) / \
(Y_count * Y_count * (Y_count - 1)))))
#saving the welch test results to both test info files
report = ['\nWelchs Test=', '','\nDate=', time.strftime("%Y%m%d%H%M%S",\
time.gmtime()), '\nFilename 1=', filename, '\nFilename 2=', filename1, \
'\nThe probablility of rejecting the Nullhypothesis (i.e. The results \
are different) is=', str(2.0 - 2 * stats.t.cdf(t_value, degrees_of_freedom))]
f = open(filename, 'a+')
f.writelines(report)
f.close()
f = open(filename1, 'a+')
f.writelines(report)
f.close()
```

Bibliography

[1] http://en.wikipedia.org/wiki/evolutionaryrobotics.

[2] Main development boinc wiki (berkley): http://boinc.berkeley.edu/trac/wiki/projectmain.

[3] C. Ampatzis. *On the evolution of autonomous time-based decision-making and communication in collective robotics.* PhD thesis, Universite Libre de Bruxelles Faculte des Sciences Appliquees, 2008.

[4] R. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72:173–215, 1995.

[5] R. Beer and J. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1, 1992.

[6] A. Christensen. Efficient neuro-evolution of hole-avoidance and phototaxis for a swarm-bot. Technical report, Universite Libre de Bruxelles Institut de Recherches Interdisciplinaires et de Developpements en Intelligence Artificielle, 2005.

[7] S. Haykin. *Neural Networks.* Prentice-Hall, 1999.

[8] D. Izzo, M. Rucinski, and C. Ampatzis. Parallel global optimisation metaheuristics using an asynchronous island-model. Trondheim, Norway, May 18-21 2009. 2009 IEEE Congress on Evolutionary Computation (IEEE CEC 2009).

[9] Dario Izzo. Formation flying for mustang. Master's thesis, Cranfield University, 2002.

[10] A. Nelson, G. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57:345–370, 2009.

[11] A. Nelson and E. Grant. Using direct competition to select for competent controllers in evolutionary robotics. *Robotics and Autonomous Systems*, 54:840–857, 2006.

[12] B. Stroustrup. *C++ Programming Language.* Addison-Wesley, 1997.

[13] J. Togelius and S. Lucas. Evolving controllers for simulated car racing. Department of Computer Science University of Essex.