Applying Evolutionary Computation to Robotics

Adrian T. Schiller
University of Minnesota, Morris
schil227@morris.umn.edu

ABSTRACT

Keywords

Evolutionary Robotics, Evolutionary Computation, Robotics, Neural Networks, Evolved Behaviors, Locomotion, Simulation

1. INTRODUCTION

In a world where things are becoming increasingly automated, robots are subject to more environments and actions. The problem with these scenarios is that it requires more complex development to account for the numerous cases that a robot may encounter. Evolutionary Computation (EC) is a very useful tool for solving complex problems, and applying EC to robots seems ideal for developing complex behaviors.

However, EC requires intensive computation for hours at a time, generating, modifying, and evaluating many candidates. This becomes implausible for a physical robot because it would require the overhead of setting it up, such as uploading the code, setting up the robot to be identical every time it is tested, running the robot in the exact same environment, and doing each task potentially thousands of times. EC for robotics is especially infeasible because it is done in real time, as opposed to traditional EC. Because of the difficulties involved and the amount of time required, it is unclear how to make EC effective in robotics.

Yet, means do exist to apply evolution to robotics, an area known as Evolutionary Robotics (ER). The purpose of this paper is to analyze what methods are used to create the evolutionary process for physical machines, the effectiveness of their outcomes, and the general feasibility of ER.

This paper will analyze three different research cases involving evolutionary robotics. By noting their similarities and differences, some idea can be generated of what ER requires. The research outcomes will demonstrate the effectiveness of the evolutionary process.

Section 2 has the background infromation, including evolutionary computation, neural networks, and the research cases this paper is based on. Section 3 covers robotic simulation and how it's used, section 4 gives the details about the evolutionary robotics. and section 5 discusses behaviors of the evolved robots. Lastly, section 6 is the conclusion.

UMM CSci Senior Seminar Conference, April 2014 Morris, MN.

2. BACKGROUND

In order to understand the complexities behind evolution and robotics, we must define the evolutionary computation, genetic algorithms, and neural networks, as well as introduce some research cases.

2.1 Evolutionary Computation

Evolutionary computation (EC) is described by Sipper [4] as solving a problem by tweaking a population of candidates and evaluating them by some quantifiable fitness. This process then repeats until a suitable candidate can be found or a set limit is reached (e.g. time). The concept can be compared to natural genetics and evolution.

Genetic Algorithms (GA), a type of EC, mimics natural selection by using a population of candidates whose success is quantified by a fitness function, and then used to create the next generation of individuals. When populating the next generation, candidates are commonly crossed-over with one another, which produces two new individuals based on the candidates. Mutation is another means of modifying a candidate, which has a random chance of occurring and slightly changes the candidate. The cross-over/mutation process repeats until the new population is created. The population is then evaluated for fitness, picks the top performing candidates, and then applies cross-over/mutation. This process repeats until a suitable candidate can be found or a limit is met (e.g time).

An example problem which we can apply GA is "Ones-Max" Suppose there is a population of p candidates which are arrays of length n whose elements are randomly set either zero or one, and the goal is to evolve a candidate array that consists of entirely ones. In the first iteration, I would apply the fitness function which would return the number of ones in an array, and then sort the population of candidates from greatest fitness to least. Then, I would choose some subset of m individuals to populate the next array of candidates (m for example, could contain the 20 best individuals). The m candidates that move on would then be used to create the next generation of candidate arrays via cross-over/mutation until p candidates are constructed. In this example, the crossover process would take two fit candidate arrays and swap parts from either array to make a new candidate; this candidate may also undergo some mu-

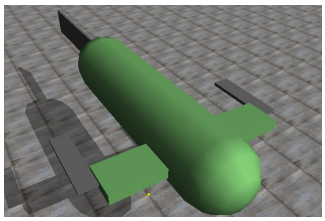
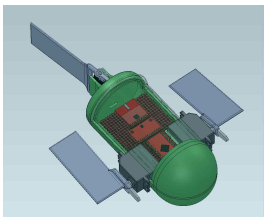



Figure 1: The station keeping robot rendered in simulation

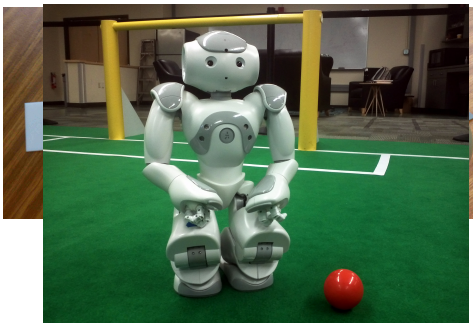



Figure 2: The Aldebaran Nao Robot

tated. This process then continues until a suitable candidate is reached, or a time/iteration limit is reached. The result would be the best candidate (i.e. the candidate with the highest fitness) which would (in this example) contain the most ones.

2.2 Neural Networks

A Neural Network (NN) is an algorithm which is based loosely on the central nervous system in biology. In EC, Artificial Neural Networks (ANNs) are used instead of NN to distinguish between a physical neural network and those artificially generated in software programs. The model is a collection of nodes ("neurons") which are connected in a network. The ANN has three general layers: input, middle, and output. The input layer is simply composed of whatever input is provided; in the case of a robot, it could be motor controllers and other sensors used by the robot. Each sensor or motor would be a separate node. The middle layer is the hidden layer where the evolutionary process occurs. From the genetic algorithm example in 2.1, instead of swapping bits in an array, the nodes of a neural network would be what is primarily swapped, changing the node network. The middle layer is not necessarily only one layer, but for the purposes of this paper it is considered as one single layer. The output layer is the result of the computation from the input layer and hidden middle layer.

2.3 Research Cases

In this paper I will compare and contrast three separate research cases which used Evolutionary Robotics. Each research study used three common methods to apply evolution to a physical robot; a simulation, an evolutionary process, and analyzing the behavior of evolved candidates.

The first study, by Moore *et al.* [2], is a aquatic robot which is tasked with maintaining a particular position while floating on water by The robot has a cylindrical body with three servo controlled fins; two flippers which have 360° movement range and a caudal (rear) fin which has 30° of movement. The robot is equipped with an inertial measurement unit (IMU) which measures linear and angular acceleration, and is able to provide a 3D coordinate of the robot's current position. The problem being solved here is for the robot to maintain a fixed position on the water while being subject to laminar flows, or water which flows in a common direction. ER was chosen for finding a solution because of the variety of directions the laminar flows could encounter the robot.

The next study, conducted by Farchy *et al.* [1], was to increase the walking speed of a humanoid robot. The robot model known as the Aldebaran Nao (Figure **??**wRobot)), is programmed by teams to play (robot) soccer in a competition called RoboCup. The process by which Farchy *et al.* applied ER was through Grounded Simulation Learning (GSL), which essentially added human guidance to the evolutionary process (more on GSL in section 4.1).

The final study, by Pretorius *et al.* [3], created a Lego Mindstorm robot which would track its own position and heading from an evolved controller (Figure **??**). Because tracking a robot without sensory input is difficult, ER was applied to create a correlation between motor movement and location/orientation. The robot consists of two motorized wheels and has two blue and purple tracking markers on the top, as well as a light sensor. The markers and light sensor were used by an overhead camera to collect position and orientation information which was used to train the controller.

3. SIMULATION

Evolution in robotics is expensive because of the overhead of running trials in real time and setting up the robot to be identical for each test. Simulation is valuable because it can explore the space of ER without requiring setup and at an accelerated rate. Simulation is the representation of characteristics or behavior of one system through the use of another system (CITE), simulation provides a means through which a robot replica can be rapidly altered and evaluated, thereby enabling the evolutionary process to occur. (Discuss state change in simulator?) However, because simulations are often imperfect, there is some measure of error by transitivity.

Transitivity in this case is defined as inaccuracies between the simulated and physical robot and the simulated/physical environment. Small inaccuracies can sometimes lead to poor performance; that is, when a candidate performs well in simulation, but performs poorly with the physical robot. An example of this would be if a simulation had assumed the surface a walking robot was moving on was a completely level plane, but in actuality it was slightly sloped; we could envision a robot which evolved to perform well in the simulation but might be unable to handle the change in slope and perhaps fall down or move more slowly. Therefore it is critical to have a simulation which has little transitivity error.

Moore *et al.* [1] evolved swimming station keeping robots with a simulator which uses the Open Dynamics Engine (ODE). The simulated environment updated every 5 ms to calculate the next state of the robot and environment. It provides a system to calculate forces applied to the simulated robot in a body of water, but does not have fluid dynamics. Instead, physical drag is applied to each of the

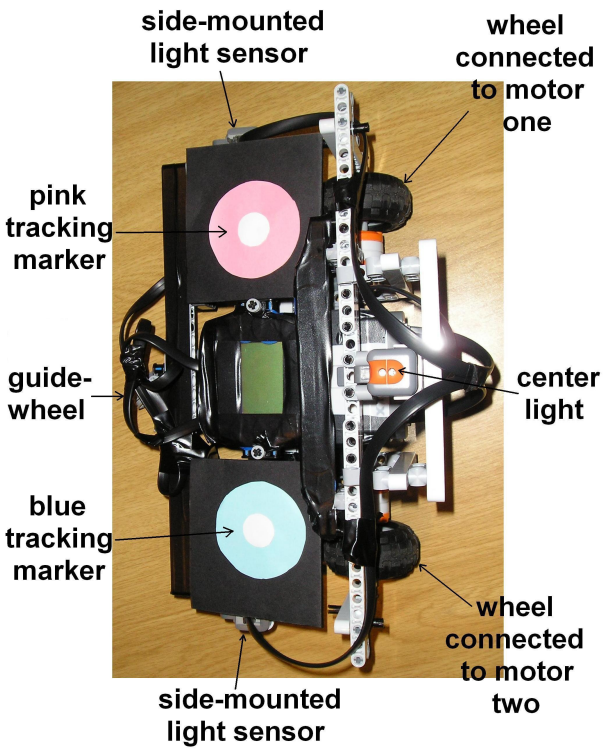


Figure 3: The position tracking robot

faces of the simulated robot. Propulsion is found to be the net force generated by each of the faces against the drag, which determine the next state of the robot. Fluid dynamics are computationally intensive and requires a lot of processing to do well. Alternatively, creating a simple model requires much less CPU processing time and therefore scales better for evolving candidates. Moore *et al.* noted that an extremely accurate simulation was not a priority because they were more interested in the evolved behaviors, that is solutions the evolved candidates create, rather than exact specifications (See Behavioral Analysis 5.1).

The robot walking code modified by Farchy *et al.* [2] used SimSpark, a simulation based on the RoboCup tournament which also uses the ODE for rigid body movement and collision detection. It is not a perfect simulation however, and lacks important features such as joint friction. Unlike the station keeping robots, transitivity errors pose more of an issue because of the threat of the physical robot being immobilized (e.g. falling over). Every 20 ms the simulator updates the state of the robot by using sensor information fed into the simulation. The robot used in the simulator is not the same model as the physical one used by Farchy *etal*, requiring approximations of key features such as the shape of the foot. However, Farchy *et al.* employed a "Grounded Simulation Learning" (GSL) algorithm, which, like the station keeping robot study, was used for finding developed behaviors and, in this case, modifying the evolutionary process from the analysis. GSL also accommodates for inaccuracies in the simulation by routinely analyzing if evolved controllers are indeed applicable to the physical robot.

The simulation used by Pretorius et. al. [3] used a more unorthodox method. As opposed to using a physical simulator (such as ODE), they chose to use artificial neural networks as the simulator. The reason the team went with this option is because a physics engine, as previously stated, contains minute inaccuracies which can affect the results. In this case, the simulator is also what is being evolved, the goal being to create a simple navigation controller which would effectively simulate the robot's location and direction. By driving the robot over a surface using arbitrary motor commands, and tracking the position and orientation with an overhead camera, they were able to extract data to be used as a testbed for the evolutionary process. In this case, error by transitivity is relative to the accuracy of capturing the position and orientation of the robot. Fortunately, the relative error was fairly small, as the observed results were within 2 cm of the physical robot's position.

In each of the previous research cases, there was a way in which the process of evaluating the robot was taken from the physical condition and placed in simulation. The simulation then enabled evolutionary robotics to avoid the overhead of physical testing and to happen at an accelerated rate.

4. NEURAL NETWORK, EXPERIMENT, AND EVOLVING PROCESS

The process of evolving candidates (addressed in section 2) was applied to each of the research robots. This section details how each case applied ER.

4.1 Station Keeping Robot

The input of the ANN for the station keeping robot [2] was the current position of the robot (x, y, z), the difference between the the robot's current position and the desired position, and the previous output. The output nodes were the oscillation of the Caudal fin, and speed of the flipper servos. The desired outcome of this neural network is to have the robot properly orient itself and maintain a fixed position based on the input.

The simulated robot was subjected to 4 different types of Laminar flows; from the front, the back, the side, and at a 45° angle from the front left. Because Moore *et al.* [2] were interested in behaviors evolved by the robot, each of these trials was a separate evolutionary process so as to not create a single candidate which had to perform well in all four. A transient period was used, which allotted 60 seconds for the robot to adjust to the particular flow. This was because early candidates would try to maintain the position immediately and have poor results; the transient period allowed the candidate to reorient itself to better maintain the position without an early penalty.

The evolutionary process had used a population of 100 candidates and evolved them for 2000 generations. Each of the four trials replicated the evolutionary process 25 times. After the 60 second adjustment phase, the simulated robot candidate was evaluated every 250 ms for the next 60 seconds. The robot's total fitness was the summation of the evaluated fitness every 250 ms;

$$\text{fitness} = \sum_{t}(10 - d_t(x, y, z))$$

where

$$d_t(x, y, z) = \begin{cases} 10, & \text{if distance}_t(x, y, z) > 10 \\ \text{distance}_t(x, y, z), & \text{otherwise} \end{cases}$$

where the distance function is how far the robot's position was from the desired location. The 10 in the fitness function is an arbitrary number to quantify the fitness of the candidate. Negative values were not counted against the fitness. The fitness is modified to have a gradient effect when the robot is close to the target location, which incentivizes continual station keeping.

4.2 Walking Humanoid Robot

The input of the ANN for the Walking Humanoid Robot selected by Farchy *et al.* consisted of 17 different parameters, including `stepPeriod`, $\texttt{amp}_{\texttt{swing}}$, `startLength`, etc. which essentially act as functions that alter multiple joints in the robot walk cycle.

The robot was evaluated in two separate trials. The first trial, `goToTarget`, gave the simulated robot several locations to walk to, dealing with several changes in direction. The fitness of `goToTarget` is:

$$\text{fitness} = \sum_{t}(\text{DistanceTraveled}_t) - \text{fallingPenalty}$$

where `DistanceTraveled` is the the length between each destination. All of the `DistanceTraveled` values are summed until the trial ends or the robot falls down. `FallingPenalty` is a penalty administered to the fitness should the robot fall before completing the trial, and is not included in the summation. The other trial is `WalkFront`, which evaluates the velocity of the robot walking forward for 15 seconds. The robots fitness for
tt WalkFront is the maximum velocity it achieves. Together, these two trials evaluate how quickly a robot can move and how stable it is.

Farchy *et al.* [1] used an algorithm slightly different from GA called Co-variance Matrix Adaptation Evolution Strategy (CMA-ES). The main difference between the two is CMA-ES tracks a co-variance matrix of parameters which perform well, thereby drawing a correlation between well performing candidates and their parameters. Instead of sorting solely by fitness like GA, it sorts by these parameters and repopulates the next generation by the upper mean of the previous generation.

One of the most interesting consepts done by Farchy *et al.* [1] and the focus of their research was the use of Grounded Simulation Learning (GSL). GSL is essentially tweaking the process of evolution as it evolves to focus on different things. In this particular case, after the first iteration of evolution the team uploaded the candidate to the physical robot and noticed that the leg swing parameter seemed to play an important role in the speed of the robot. The next iteration was then adjusted to have greater variation in the swing parameter to be used in the CMA-ES algorithm.

4.3 Position Tracking Robot

To train the position tracking robot's ANN, the robot was issued commands [3]. These commands consisted of three things: motor speed for the both motors, and the execution time for the command in milliseconds. The commands randomly generated the motor speed, which ranged up to 50% of the maximum motor speed, the a direction of the motor, and a time ranging from .5 to 3 seconds. Although random, the seed was bias to have the motors go in the same direction at an equal speed (straight movement) 30% of the time, as well as both motor speeds to equal 0 (stopping the robot) 20% of the time. This was because Pretorius *et al.* felt that these would be common commands, and should be represented well in the simulation.

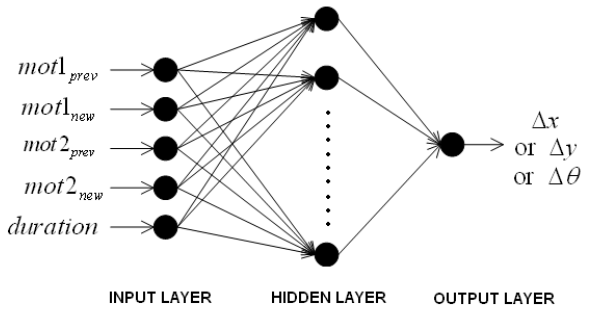


Figure 4: The ANN for the position tracking controller

Pretorius *et al.* used three different simple ANNs to track the robot's position. The input to each of the neural networks consisted of the two motor speeds before receiving the command, the two motor speeds from the current command, and the length of time for the current command. The purpose of using the previous motor command was to account for positive and negative acceleration the robot underwent going from one command to the next; the neural network would have to incorporate that change in order to make an accurate account of it's orientation. The output of the ANNs was either the robot's x-coordinate, y-coordinate, or angle. Figure 4 is a visual representation of the ANNs.

The ANNs were then evolved using a GA. The GA parameters included a 80% crossover probability and a 5% mutation possibility. The population size was 250 candidates, which were evolved for 15,000 generations. This took approximately 12 hours for each of the three ANNs. The fitness function was the Mean Squared Error (MSE), which measured the accuracy of a given candidate ANN configuration. The MSE is defined as:

$$\text{fitness} = \frac{1}{N}\sum_{p=1}^{N}\sum_{i=1}^{O}(t_{pi} - a_{pi})^2$$

Where N is the size of the testbed, O is the number of outputs from the ANN, t is the expected output, and a is the actual output. Both t and a are are subject to some particular test p in the testbed and for either the x-coordinate, y-coordinate, or angle, i.

In addition to evolving the ANN controllers, Pretorius *et al.* evolved a navigational controller, a set of commands, to drive the robot around a 3 by 3 rectangular grid using the evolved ANNs. The objective was to have the robot start in the center left spot and drive counter clockwise around the grid to end back up in the starting spot, while avoiding the middle rectangle, staying on the grid, and visiting each space in a procedural manner. If the navigation controller violated any of those requirements, it would be heavily penalized. The fitness function was:

$$\text{fitness} = N_{max}^2 - \frac{C_{nogain}}{10}$$

Where N_{max} is the absolute number of rectangles traversed while not entering forbidden areas and C_{nogain} is the number of commands which did not cause the robot to advance to the next rectangle.

In order to effectively evolve the navigation controller candidates, cross-over mutation was modified to swap parts that were relatively close to the same part in the grid. For example when crossing two candidates, the part of their structure which issued a 'turn' command at the top left square of the grid would be marked as the swap point. This resulted in more consistent candidates and more effective evolution.

5. RESULTS AND BEHAVIORAL ANALYSIS

Evolved code may be confusing or difficult to alter. By observing evolved behaviors, it is possible to learn the general solution and then create code to mimic that solution. This is not a necessary step, but it can provide some insight as to what the evolved candidate is doing to be successful.

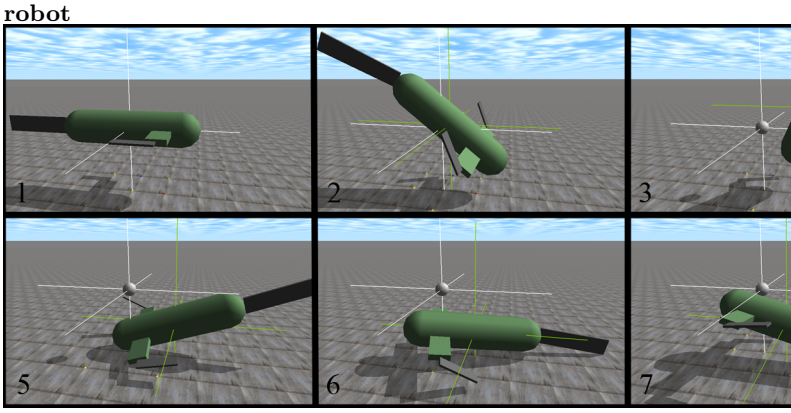
5.1 Station Keeping Robot

The evolutionary solutions for the station keeping robot in [2] were both unforeseen and effective. Depending on the trial, the evolved candidates would range from standard locomotion to complex maneuvers. When the flow was coming from either the front or at a 45° angle to the front left, the simulated robot would swim forward or swim forward while listing to the left. Because of the design of the robot, it is somewhat difficult for it to turn and not move too far away from the starting point. This would make station keeping especially difficult when the flow was coming from the back or side, the robot would drift away as it would turn, resulting in a poor fitness. so instead, when the flow was coming from directly behind the robot, it used a complex maneuver to flip tail-over-head to face the flow and then propelled itself forward to maintain its position (http://y2u.be/UufbnEGFwV4). Figure 5 shows the behavior. This works well because the robot floats, and the flipping movement doesn't cause a change in it's height in the water. The most challenging trial was when the flow came at a 90° angle; because reorienting to such an angle was evidently a major challenge, the evolved behavior incorporated a flipping motion combined with a rolling the body to avoid turning.

Moore *et al.* [2] compared early candidates with final candidates to contrast in evolved behaviors. When the flow was coming from behind, an early candidate selected from the 25^{th} generation was pushed by the current straight out of the 'rewards zone' (the desired station keeping location). It didn't actually start swimming until 60 seconds into the trail, and was unable to make any progress in the allotted time. In contrast, the final candidate immediately reacts and is able to orient (flip) itself and gain against the flow into the 'rewards zone'.

Moore *et al.* [2] noted that one of the most impacting changes was to allow a setup phase. Allowing the setup phase allowed the candidate to have a poor initial fitness as long as it was able to attain a good fitness later. This was a valuable choice because time was superfluous, and it relieved pressure to perform well immediately. This case demonstrates the power of the fitness function; the function guides the simulation and results will directly correlate to what the function is optimizing.

Figure 5: Images from the simulation which showed the evolved behavior when the flow was behind the robot



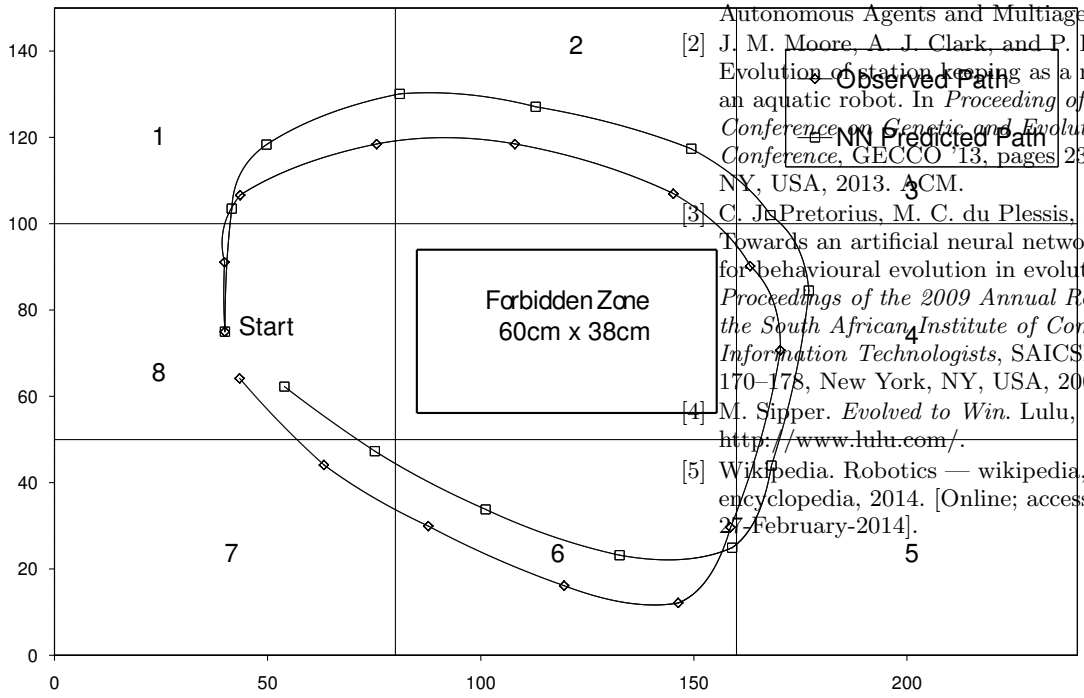
5.2 Walking Humanoid Robot

The results of the ER for the walking robot [1] had improved the walking speed, and the improvement was statistically significant. Before evolution, the walking speed of the robot was 11.9 cm/s. After the first iteration of evolution the maximum walking speed increased to 13.2 cm/s, however the robot was not as stable as the original. The next iteration used the same parameters as the first (increasing the leg swing) and the walking speed had increased to 14.6 cm/s. The following GSL focused more on stabilizing the robot, and a parameter set was found such that the robot was able to move at 15.9 cm/s and not fall down at all. Initially the maximum length of the robot's step was reduced by 1/3 because the parameters the robot used were unstable in the real tests but worked fine in simulation. It was found that by the forth iteration this was no longer an issue, and the step size was restored to its maximum. At the maximum step size, the original velocity of 11.9 cm/s increased to 13.5 cm/s, and the final candidate robot was able to move at 17.1 cm/s, a 26.7% increase.

5.3 Position Tracking Robot

The position tracking robot's [3] three ANN had evolved to be competent at tracking the robots coordinates and angle. The top x-coordinate, y-coordinate, and angle ANNs were tested with 50 commands to test their accuracy, the results are located in <FIGURE >. To validate the experiment, Pretorius *et al.* tested the navigation controller evolved with the ANNs. 6 shows the results of the navigational test controller, which used 13 commands to navigate around the grid. Pretorius *et al.* were satisfied with the results, noting that they were able to accomplish the specified task without any manual robotic programming. The simulated and actual paths were reasonably close to one another, meaning that the ANNs were able to achieve a relatively accurate account of the position and orientation of the robot. The error between the actual and simulated run was expected because slight errors tended to compound, i.e. after each command the simulator would be slightly inaccurate plus the sum of the previous command's inaccuracies.

Figure 6: Predicted and actual robot paths from the navigation controller, which used 13 commands



6. CONCLUSIONS

These research cases each demonstrate successful applications of ER. We can draw the conclusion that not only is evolutionary robotics feasible, it can in fact quite effective. Simulation provided an important platform upon which evolution could take place. That platform can be a physical representation of the robot and the environment, or by a sizable testbed as seen in the robot developed by Pretorius *et al*. [3]. Each research case used evolutionary computation to evolve neural network controllers, most likely because neural networks have been shown to be an effective tool for developing relationships from sensors and motors.

The robots from the research cases tended to diversify when it came to observed behaviors. The station keeping robot developed by Moore *et al*. [2] evolved many interesting behaviors, however no single candidate was made to excel in all of the trials. This was because Moore *et al*. wanted the simulated robot to master a given trial to gain some understanding of what an effective action would be, and then encode that functionality on the robot. In contrast, the walking robot developed by Farchy *et al*. [1] and the coordinate tracking robot by Pretorius *et al*. [3] directly used their candidate solution in the physical robot.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone. Humanoid robots learning to walk faster: From the real world to simulation and back. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 39–46, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[2] J. M. Moore, A. J. Clark, and P. K. McKinley. Evolution of station keeping as a response to flows in an aquatic robot. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 239–246, New York, NY, USA, 2013. ACM.

[3] C. JuPretorius, M. C. du Plessis, and C. B. Cilliers. Towards an artificial neural network-based simulator for behavioural evolution in evolutionary robotics. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '09, pages 170–178, New York, NY, USA, 2009. ACM.

[4] M. Sipper. *Evolved to Win*. Lulu, 2011. available at http://www.lulu.com/.

[5] Wikipedia. Robotics — wikipedia, the free encyclopedia, 2014. [Online; accessed 27-February-2014].