Applying Evolutionary Computation to Robotics

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

ABSTRACT

Evolutionary computation (EC) is a useful tool for solving difficult problems which have a non-obvious solution. However, because robots resides in the physical plane and cannot be evolved rapidly, it is questionable how useful EC is in the field of robotics. By studying three research cases which apply EC to robotics, a consensus can be reached about what methods are required to do so. It becomes apparent that a simulation is required to run EC rapidly, and the results of the cases show EC applied to robotics can be effective.

Keywords

Evolutionary robotics, evolutionary computation, robotics, neural networks, evolved behaviors, locomotion, simulation

1. INTRODUCTION

As robots become increasingly automated, they are used in a wider variety of environments and expected to perform a wider variety of actions. The problem with this is that it requires more human effort to encode the robot with the proper behaviors for the numerous cases which the 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 impractical for a physical robot because it would require the overhead of uploading the code, setting up the robot and its environment to be identical every time it is tested, and doing each of these tasks thousands of times. EC for robotics is especially challenging because it is done in real time, as opposed to traditional EC which can evaluate many candidates relatively quickly on a computer. Because of the precision and 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). This paper will analyze three different research cases involving ER. Each case has a robot tasked with solving a particular problem; maintaining a position on a body of water, increasing walking speed, and tracking their coordinates without sensory input. By analyzing the research cases, it was found that EC can be applied to robots by using a simulation, and the outcome have shown that ER is effective.

Section 2 has background information on evolutionary computation, neural networks, and the research cases this paper uses. Section 3 covers robotic simulation and how it is used in these research cases, Section 4 gives the details about the evolutionary robotics for each of the research cases, and Section 5 discusses results and behaviors the evolved robots developed. Section 6 presents the conclusion of this paper.

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

2. BACKGROUND

In order to understand the complexities of evolution and robotics, we must define evolutionary computation with genetic algorithms, and neural networks. Three research cases will be introduced as well.

2.1 Evolutionary Computation with Genetic Algorithms

Evolutionary computation (EC) is described by Sipper [5] as solving a problem by tweaking a population of candidates and evaluating them using a quantifiable measure of success. A fitness function serves the purpose of evaluating a candidate's ability; the candidates which perform well are used to create the next generation of individuals. This process repeats until a suitable candidate can be found or a limit is met. When populating the next generation, the top candidates are crossed-over with one another, which produces two new individuals based on the candidates. Mutation is another means of modifying a candidate, which is the random chance of slightly changing the candidate. Candidates are crossed-over/mutated until the new population is created.

Genetic Algorithms (GA) [7], a type of EC, represents a candidate as a bit string. In GA, performing cross-over means that one or several pairs of indices are chosen to be swapped between two candidate bit strings. Likewise, mutation would flip a random bit in the candidate bit string. A sample problem which GA can be applied to is "OneMax". The purpose of "OneMax" is to evolve a binary array to consist entirely of ones. It starts with a population of p candidates which are arrays of length n whose elements are randomly set to either zero or one. In the first iteration, the fitness function would be applied which would return the number of ones in an array. Then, some subset of the population, m, is chosen to populate the next array

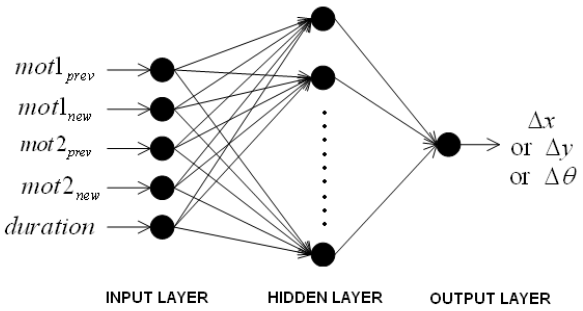


Figure 1: The ANN for the position tracking controller. Taken from [4]

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 takes two fit candidate arrays and swaps parts from the arrays to make two new candidate arrays. These candidates may also undergo some mutation, such as flipping a random bit in the array. This process then continues until a suitable candidate is reached, or a time/iteration limit is reached. The result is the candidate with the highest fitness; in this example the candidate with the most ones.

2.2 Neural Networks

Two of the three research cases presented in this paper use artificial neural networks ([3], [4]). An Artificial Neural Network (ANN) [6] is an algorithm which is based loosely on the central nervous system in biology. The network is a connected collection of nodes ("neurons"). The network is composed of weighted vertices (synapses in the biology analogy) between all the nodes, which act as a control structure to transform the input to the output. The goal of the network is to develop an approximated relationship between a given input and a desired output. Figure 1 is an example of an ANN.

The ANN has three general layers: input, middle, and output. The middle layer is not necessarily only one layer, but for the purposes of this paper it is considered a single layer. The input layer is composed of whatever input is provided. In the case of a robot, it could be motor controllers and other sensors, each of which would be 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, weights of the network would be swapped or altered. The output layer is the result of the weighted inputs. For example, there could be an ANN which takes in the velocity of a car and how long it has driven as its input, and the output could be a coordinate on the xy-plane of its location, calculated from a mathematical formula developed in the hidden layer.

2.3 Research Cases

This paper compares and contrasts three separate research cases which used Evolutionary Robotics (ER). Each research study used three common approaches to apply evolution to find a solution for a physical robot; a simulation, an evolutionary process, and behavioral analysis of evolved candidates.

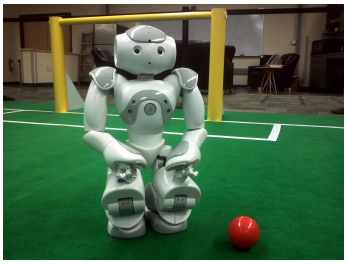


Figure 3: The Aldebaran Nao Robot, programmed to play soccer

The first study presented here, done by Moore *et al.* [3], is a simple floating aquatic robot which is tasked with maintaining a particular position while being subjected to water flowing in a single direction, also known as laminar flows. The robot has a cylindrical body with three servo-controlled fins; two flippers which have 360° of circular movement range and a caudal (rear) fin which has 30° of movement. Figure 2 shows the robot in simulated and physical form. 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. ER was chosen for finding a solution because of the numerous interactions between the robot and the laminar flows.

The next study, conducted by Farchy *et al.* [1], was to increase the walking speed of the humanoid robot Aldebaran Nao (Figure 3). This robot 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 added human guidance to the evolutionary process (more on GSL in section 4.2).

The final study, by Pretorius *et al.* [4], created a Lego Mindstorm robot which would track its own position and heading from an evolved controller, without using any external sensors. This controller would be able to calculate where the robot is located as it moves, only using the motor's speed, direction, and running time. The robot (Figure 4) 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, and pair it with commands given to the robot. This data was used as a testbed to train the controller. This is a difficult problem because without sensory input, tracking a position must be done using internal functions, which tend to be complex because of various accelerations the robot undergoes when moving, and inaccurate due to friction and imprecise movement (slippage, low battery, etc.). ER was chosen to create a correlation between motor movement and location/orientation.

3. SIMULATION

Physical 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

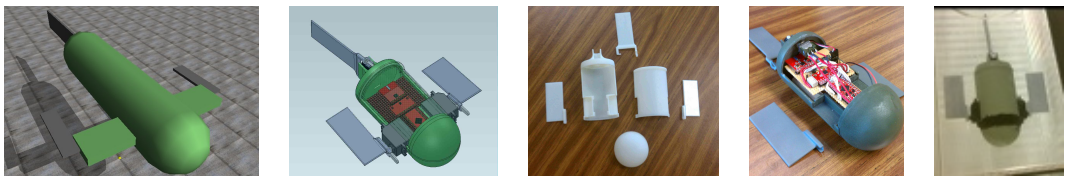


Figure 2: The station keeping robot rendered in simulation as well as in it's physical form. Taken from [3]

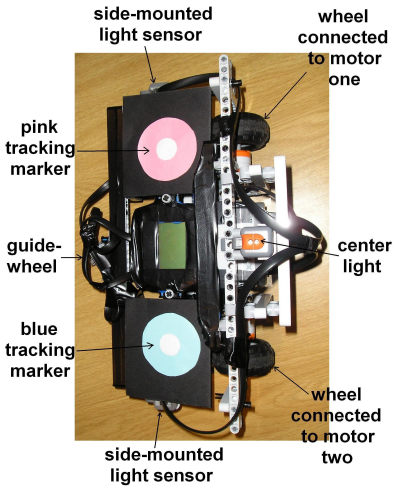


Figure 4: The position tracking robot. Taken from [4]

because it can explore the space of ER at an accelerated rate without requiring setup. Simulation is the representation of characteristics or behaviors of one system through the use of another system. Simulation provides a means by which a simulated robot can be rapidly altered and evaluated, thereby enabling the evolutionary process to proceed efficiently and quickly.

However, because simulations are often imperfect representations of the physical space, there is some accumulation of error by transitivity [1]. Even small inaccuracies can sometimes lead to poor performance where a candidate performs well in simulation, but performs poorly with the physical robot. Consider, for example, if a simulation had assumed the surface a walking robot was moving on was a completely level plane. If 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. Therefore it is critical to have a simulation which has little transitivity error when applying ER.

Moore *et al.* [3] evolved floating 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 take into account fluid dynamics. Instead, physical drag is applied to each of the 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 require a lot of processing to do accurately. 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 Section 5.1).

The robot walking code modified by Farchy *et al.* [1] 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 robot, transitivity errors pose more of an issue because they can off balance the robot and potentially immobilize it by causing it to falling down. 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 *et al.*, 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 based on the analysis of intermediate candidates. 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. [4] used a more unorthodox method. As opposed to using a physics 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, the error caused by transitivity is relative to how accurately the camera captured the position and orientation of the robot. Fortunately, the relative error was fairly small because the captured results were within 2 cm of the physical robot's position.

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

ronment into a simulation, it allows ER to be feasibly done in a reasonable amount of time.

4. EVOLUTIONARY 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, including population size, number of generations, how they implemented neural networks, fitness functions and evaluation methods.

4.1 Station Keeping Robot

The input of the ANN for the station keeping robot [3] was the current position of the robot in three dimensional coordinates, (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 (90° from the front), and at a 45° angle from the front left. Because Moore *et al*. [3] 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 immediately maintain the position 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 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

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

and 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. 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 walking robot [1] had 17 different parameters, including `stepPeriod`, $\texttt{amp}_{\texttt{swing}}$, `startLength`, etc. which act as functions that alter multiple joints in the robot walk cycle. Farchy *et al*. applied ER to optimize the parameters to increase the standard walking speed. It is worth noting that they did not use an ANN, and instead evolved the parameter sets with different values using a GA-like algorithm.

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 to 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. The other trial is `WalkFront`, which evaluates the velocity of the robot walking forward for 15 seconds. The robots fitness for `WalkFront` is the maximum velocity it achieves. Together, these two trials evaluate how quickly a robot can move and how stable it is, evolving a both fast and stable robot.

One of the most interesting concepts done by Farchy *et al*. [1] and the focus of their research was the use of Grounded Simulation Learning (GSL). GSL is composed of two main parts: grounding and guidance. Grounding refers to making the simulation's behavior match the physical robots behavior; this reduces problems with transitivity. Guidance refers to human interaction in the simulation to make strategic adjustments in the evolutionary process. GSL was applied after each iteration to focus on evolving specific features, such as taking longer strides or improving better balance. For example, 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 evolutionary algorithm.

4.3 Position Tracking Robot

To train the position tracking robot's ANN, the robot was issued commands [4] consisting of three elements: motor speeds, the directions of each of the motors, and the execution time for the command in milliseconds. The motor speed was randomly generated in the range from 0 to 50% of it's maximum, and the time ranging from .5 to 3 seconds. Although random, there 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 emphasized in the simulation.

Pretorius *et al*. used three different simple ANNs to track the robot's position. Similar to how the station keeping robot evolved four different candidates for each of the laminar flows, using three different networks removed the demand for a single network to perform well in tracking both coordinates and the orientation. 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 its orientation. The output of each of the ANNs was either the robot's x-coordinate, y-coordinate, or angle. Figure 1 is a visual representation of the ANNs.

The ANNs were then evolved using a GA. The GA pa-

rameters 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 (i.e. the collected data from arbitrary motor commands and their physical position, as gathered from the overhead camera), 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, which was 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 which the robot would navigate. 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

This section discusses results found from each of the research cases. This includes some observed behaviors with the swimming and walking robots, which shed insight on what candidates evolved to do to find a solution to their problem.

5.1 Station Keeping Robot

The evolutionary solutions for the station keeping robot in [3] 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 [2]. Figure 5 shows the behavior. This works well because the robot floats, and the flipping movement didn't cause a change in its height in the water. The most challenging trial was when the flow came at a 90° angle; because reorienting to such an angle was a significant challenge, the evolved behavior incorporated a flipping motion combined with a rolling the body to avoid turning.

Figure 6: The accuracies of the evolved coordinate tracking ANN. Taken from [4]

NN Simulator	Final MSE	Average absolute error
change in angle	26.412	3.585 degrees
change in y-coordinate	12.909	2.143 cm
change in x-coordinate	18.559	2.782 cm

Moore *et al.* 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 did not actually start swimming until 60 seconds into the 120 second trial, 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.* noted that one of the most impacting changes was to allow a setup phase. 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 decision because the speed of maintaining the position was unimportant, and by allowing more time for candidates to orient themselves relieved pressure to perform well immediately. By using a well constructed fitness function, the ER process is shown to be very effective in this case.

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, however it was more prone to falling. 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 without loss of stability. 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 evolved parameter set enabled the robot to move at 17.1 cm/s, a 26.7% increase.

5.3 Position Tracking Robot

Figure 5: Images from the simulation which showed the evolved behavior when the flow was behind the robot. Taken from [3]

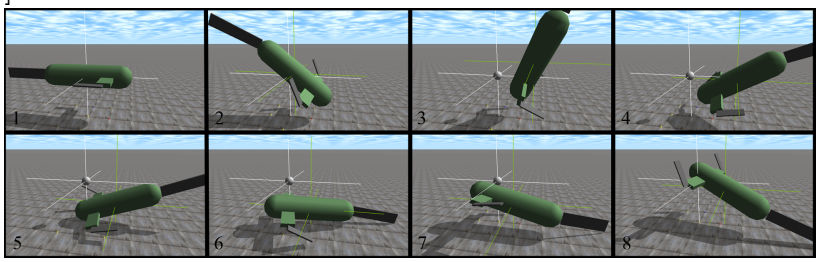
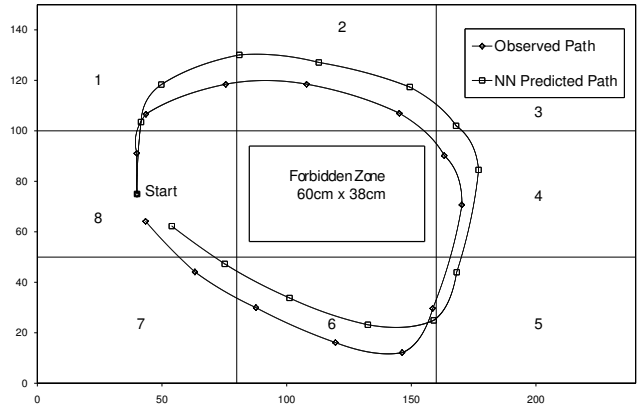


Figure 7: Predicted and actual robot paths from the navigation controller, which used 13 commands. Taken from [4]



The position tracking robot's [4] three ANN had evolved to be competent at tracking the robots coordinate and angle. The top x-coordinate, y-coordinate, and angle ANNs were tested with 50 commands determine their accuracy, the results are located in Figure 6. To validate the experiment, Pretorius *et al.* tested the navigation controller evolved with the ANNs. Figure 7 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 manually programming the robot. 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.

6. CONCLUSIONS

These research cases each demonstrate successful applications of ER. We can draw the conclusion that evolutionary robotics can be 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.* [4]. Despite the problems presented by each of the research cases, each were able to find a suitable candidate solution, be it through ANN or parameter sets. Despite the diversity of environments and interactions a robot may be subjected to, if the robot can be represented in simulation and have a well defined fitness function, ER is a viable option.

7. ACKNOWLEDGMENTS

Thanks to Nic McPhee, Elena Machkasova, and Alex Jarvis for their invaluable feedback.

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. Trial 2: Evolved individual, 2013. [Online; accessed 26-March-2014].

[3] 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.

[4] C. J. Pretorius, 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.

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

[6] Wikipedia. Artificial neural network — wikipedia, the free encyclopedia, 2014. [Online; accessed 26-March-2014].

[7] Wikipedia. Genetic algorithm — wikipedia, the free encyclopedia, 2014. [Online; accessed 7-March-2014].