EVOLVING ROBOTIC DESIRES: A NEW APPROACH TO BRIDGING THE REALITY GAP

Russell Toris
Dickinson College
torisr@dickinson.edu

ABSTRACT

Since the emergence of the field of Evolutionary Robotics, new breakthroughs have been made to further its development and prove its effectiveness. While from a purely engineering standpoint the modeling of biological evolutionary phenomena may not seem to be the most efficient means of implementation, it has proven to be able to provide solutions to interesting problems within the robotics world. The approach of evolving autonomous controllers for robots has had proven benefits when compared to a more traditional hand-coded approach.

Like most fields of research, Evolutionary Robotics contains its own set of problems. One such problem involves the use of simulators to speed up the evolutionary processes. When transferring the robotic controller from the simulation to the physical robot it tends to perform poorly on a given task. This issue is referred to as the reality gap problem. In this paper, a new approach to bridging the reality gap is presented and explored. The goal is to evolve a robotic controller that gains correct desires based on its current state and uses reinforcement learning to perform actions that achieve such desires. By doing so, the goal is to have a robotic controller adapt to uncertainties and perturbations within the real world once transferred from simulation.

KEY WORDS
Evolutionary Robotics, Q-Learning, Genetic Algorithms, Reinforcement Learning

1. Introduction

The area of Evolutionary Robotics is a relatively new research field. The general approach is to take hundreds of random robot controllers (a software program used to govern the actions of a robot based on its sensor readings), evaluate their performance (also known as their fitness), and reproduce the best performing ones over a few hundred generations. Typically, such models incorporate the use of a genetic algorithm. This type of algorithm attempts to make use of natural evolutionary phenomena such as survival of the fittest [1]. The idea of using evolutionary models to build autonomous controllers for robots has several benefits over a traditional hand-coded version. For example, problems contain conditions such as the lack of a known solution, dynamic environments, and various levels of uncertainty. Such properties make it nearly impossible to design an effective hand-coded controller. In these cases, evolutionary techniques have had proven benefits when compared to a more traditional approach [2].

Since its early stages, researchers in evolutionary robotics have encountered problems that must be overcome. One widely known problem is computational expense. Evolution of a robot controller, even a controller for a simple task, usually requires hundreds of generations. Each of these generations typically has a population of over a hundred individuals. If each trial takes around sixty seconds, completing a few hundred generations with one physical robot could take weeks. As tasks get more complicated, evolutionary techniques with a single physical robot could take years or lifetimes to complete. Often times, it is impractical or financially impossible to solve this problem by using multiple robots. To overcome this, researchers have resorted to the use of simulators to conduct their trials. Simulators provide the means to run trials at a faster rate and allow multiple trials to be run in parallel.

This may sound like a simple and obvious solution to the problem; however, it is nearly impossible to recreate all aspects of the real world in simulation. No matter how much effort is put into creating realistic simulations, the real world has too much unpredictability, variability, and detail to be accurately modeled in simulation. Robot controllers that are evolved in simulation will naturally take advantage of the properties and certainties that exist in the simulator and not in the real world. Thus, even the fittest robots in simulation can fail when applied to physical hardware. Many researchers refer to the difficulty of evolving controllers that perform just as well in reality as compared to in simulation as the reality-gap problem [3], [4], [5]. This paper will focus on examining and testing a method for closing this gap.

2. Background

Several approaches have been proposed to bridge the reality-gap. One such method is the radical envelope-of-noise [3]. This method attempts to place a reasonably high amount of noise onto the simulation sensors while running the evolution trials. The idea is that any robot controller that performs with high fitness under such conditions should have the ability to deal with variability in the real world. The robot will not become dependent on artifacts of the simulation because they are masked by the noise.

An additional approach is to incorporate an anticipation model into the robot's controller [4]. During the simulation, the robot learns and stores information about its actions in the simulated environment. This information is then used when assessing its actions and their effects within the real world. The amount of error can be viewed by comparing its actions and effects in simulation against those in the real world. This information can then be used to provide correction criteria to the controller when determining what actions to make, effectively anticipating any uncertainties and work as an error estimator. The controllers use this error estimation and attempt to adjust themselves accordingly while running in the real world.

Another approach is to evaluate and promote "transferable controllers" [5]. The idea proposed here is to run a traditional genetic algorithm experiment and evaluate fitness accordingly. However, an additional criterion is used to look for novelty and diversity within the simulated robots. During the experiment, controllers that are considered diverse are chosen and run out of simulation on a physical robot. A comparison is then made between their performance in the real world and their performance in the simulator. Real world robots that perform similarly to their simulated counterparts are promoted to the next generation. This method evolves highly transferable controllers and rewards those that act similarly within the real world.

The above examples are just a few of the numerous attempts to address the reality gap problem. While each has shown positive results in their own manner, the research discussed in this paper has been based on a new attempt to bridge the differences between the simulated and real world environments.

3. Proposed Solution

The idea being investigated is to incorporate an evolved neural network that produces a set of *desires* based on its current state. These desires and the current state information (e.g. sensory values) are then fed into another module that will use reinforcement learning. The hope is that the agent will evolve to adapt to the differences between the real world and simulated environments.

Each robot controller will use two separate modules to determine its actions. Essentially, the controller will take the state information from the robot and ultimately produce an action as its output. These actions can be defined based on the specific problem at hand. For example, when dealing with robotics, the action will typically be a set of motor commands. The two modules that will be used will be referred to as the desires network and the action module. An abstract visualization is depicted in Figure 1.

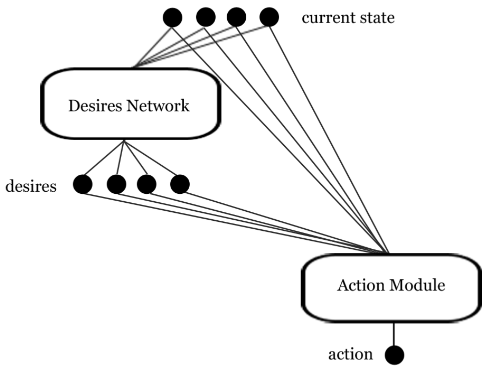

Figure 1: The basic outline and integration of the desires network and action module.

3.1 The Desires Network

The desires network will use a feed forward neural network [6]. The input to this neural network is the current state information gathered from the agent. Therefore, there will be enough neurons to represent each of the associated values. This network will have the same number of outputs as inputs. The outputs represent the *desires* for the agent. The network will also contain a single hidden layer with additional neurons. The purpose of this neural network is to have the controller express what its next state should look like based on the current state. The weights of this network are represented by the *genes* of the evolutionary controller, and thus are what will be evolved. Therefore, the idea is to have evolution produce individuals that produce correct desires.

3.2 The Action Module

The current state information and its corresponding desires are then fed into the action module. This module ultimately produces a single action to take. Throughout the agent’s lifetime, the module will be modified and trained using reinforcement learning. The general idea behind reinforcement learning is to provide the module with feedback (good or bad) and have it adjust itself accordingly [7], [8], [9], [10]. Furthermore, once the current state information and desires are fed into this reinforcement learning model, a single action is returned. Therefore, the module will be given positive feedback when it causes an action that produces a state that satisfies its desires. The proposed feedback (referred to as a reinforcement signal) is defined to be a distance measurement between the result state and the previous desires.

Throughout the agent’s lifetime, the action module will be trained and modified using this reinforcement signal. It is important to note that this reinforcement will be training the action module to produce actions that will result in a state that is comparable to its desires that were produced from the desires network. Additionally, depending on the desires network, the actions produced by the action module could be completely wrong for the purposes of the problem the controller is attempting to solve. Especially in early generations, many controllers will have harmful or incorrect desires. The fitness function defined within the experiment will assign relatively low fitness to these individuals and therefore will not allow them to reproduce into future generations. Thus, over time controllers will evolve to aspire desires that are appropriate to solve the problem.

3.3 Q-Learning

While the general approach is to use reinforcement learning, there are several different variations. The proposed type of reinforcement learning is Q-learning [10], [11]. This approach attempts to determine and provide outputs that will maximize the reinforcement signal (called the Q-function).

Each controller will have an implementation of the two modules. The desires network will be encoded within its genes and passed on to future generations (with modifications such as random mutation). The action module will use learning and thus the learned information will not be passed on to future generations. As with natural models of evolution, what we learn in life is not directly passed on. At the beginning of each trial, the controller will be given a standard default action module that it will modify throughout its lifetime using reinforcement learning.

4. Experimental Proof of Concept

The first major task in the research project was to investigate a proof of concept. This important step is necessary to verify components of the experiment in small increments before testing the large-scale experiment. Moreover, robotic experiments can take several days to run. With so many variations possible in the experiment parameters (e.g. mutation rates, gamma values for Q-learning, ext.), it is unrealistic to go straight to the large-scale trials. Therefore, the decision was made to test out the concept of using the desires network and action module on an agent in the *SimpleWorld*.

4.1 The *SimpleWorld* Model

The SimpleWorld is a discrete grid-world model that is part of the dLife software package [12]. The world is a two-dimensional grid (such as the one depicted in Figure 2) with random assignments of food, poison, obstacles, and predators. A single agent is then placed in the center of the grid and has three possible movement actions that it can take: turn left, turn right, and move forward.

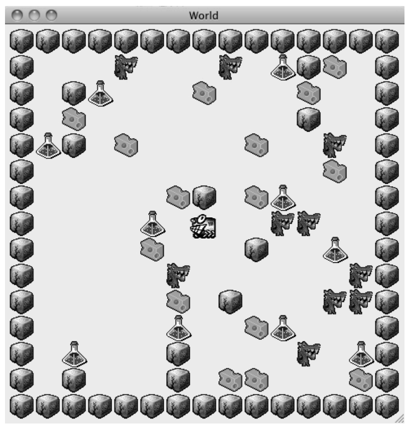

Figure 2: An example of a *SimpleWorld*.

The agent is given an initial amount of health, or life force, which determines if the agent is considered alive or dead (a positive life force corresponds to being alive). With each action, the agent will lose one life force point, when a poison piece is consumed or the agent runs into an obstacle or predator it will lose a predetermined constant amount of life force, and when food is consumed the life force will increase. Moreover, each agent has four sensory inputs: front color, right color, left color, and smell. The color and smell outputs of each object in the world are listed in Table 1.

Table 1: Color and smell outputs for objects located in the *SimpleWorld*.

Object	Color	Smell
Food	*green*	*sweet*
Poison	*green*	*sour*
Predator	*red*	*sour*
Obstacle	*red*	*none*

Each smell and color sensor has a customizable range. Therefore, the color sensor for each direction is the first color it sees in that direction or *none* if no color is within range. For the smell sensor reading, the agent will look for all smells within the radius of itself and pick one of the detected smells (if any) at random. An important note is that both poison and food have a color value of *green*; however, they have different smell values. Due to the fact that the smells are random if they are equidistant from the agent, this creates a sense of non-determinism within the world.

The goal of the agent is to move around the world for as long as it can before running out of life force. Each trial can be run without a graphical user interface to train an agent quickly in the world.

Two separate experiments were used as a proof of concept. The first experiment contained only the action module of the proposed research project. This required an implementation of Q-learning (a type of reinforcement learning) to be constructed that was then used in a

SimpleWorld environment. The second experiment was to test the desires network and action module pairing in an evolutionary experiment. For each experiment, a smell range of one and color range of three were used.

4.2 The Q-Learning Experiment

The first major task for this experiment was to create a valid implementation of Q-learning that could be used in a variety of applications. Q-learning works by finding and executing the *best action* across all possible actions for the current state. In the *SimpleWorld*, the actions can be any of the possible movements, and the state is the combination of all the sensor readings. Each action is ranked with a value referred to as the Q-Value. This value is changed over time based on how well that action satisfies the reinforcement signal [10], [11].

In addition to the states and actions, a Q-learning algorithm requires the implementation of an exploration function. The purpose of this function is to take in both the Q-value for the current state-action pairing and the number of times the current state-action pairing has been observed as parameters and return a single value. The action that maximizes this function during an iteration of the algorithm is chosen. For the purposes of the *SimpleWorld* experiment, the exploration function is seen in equation 1. This was an adaptation of an example presented by Russell and Norvig [10].

$$f(q,n) = \begin{cases} R, & n < N \\ q, & \text{otherwise} \end{cases} \qquad (1)$$

In this function, q is the Q-value for the current state-action pairing, n is the number of times the state-action pairing has been observed, R is the maximum reward possible in any state, and N is a cutoff. The purpose of this function is to force the Q-function to explore all possible actions on a given state at least N times [10]. For the experiment, a cutoff of 10 was used.

During the experiment, an iteration of the algorithm is run during each step to determine which action to take based on its current state. The reward of the action (its reinforcement signal) is the difference between the life force before and after the action is taken. Thus, a loss in life force resulted in a negative reward, and a gain in life force (i.e. it ate a piece of food) resulted in a positive reward.

Once a valid Q-learning algorithm was implemented, the experiment itself could be designed. The goal of the experiment was to verify that the Q-learning implementation worked on the *SimpleWorld* model. This experiment ran a single agent through 35,000 randomized *SimpleWorld* instances where a new instance was created after the agent died. The number of steps the agent lived in each world was recorded. If the agent successfully learned information about the world, it should start to live longer on average as the experiment progresses. It is important to note that no evolution was used in this experiment; the same agent is being run through thousands of *SimpleWorld* environments. Throughout its lifetime, the agent should learn how to move around sensibly in the world.

4.3 Q-Learning Results

After completing 100 trials of the Q-learning experiment, results were consistently positive. Agents showed clear signs of improvement during later worlds as opposed to earlier worlds. Table 2 summarizes the average lifespan of the 1st world and 35,000th world across the 100 runs of the experiment.

Table 2: The average lifespan of the 1st and 35,000th world of the agents across 100 runs.

World	Average Lifespan	Standard Deviation
1st	24.84	10.11
35,000th	64.88	26.19

In order to compare the means, a paired t-test was performed to test if the Q-learning model helped to improve the agent's performance in the *SimpleWorld*. The appropriate null and alternative hypotheses are shown in equation 2 where μ_D is the mean of the differences between lifespans from the 1st world and the 35,000th world.

$$\begin{aligned} H_0 &: \mu_D = 0 \\ H_1 &: \mu_D < 0 \end{aligned} \qquad (2)$$

The mean of the differences between lifespans from the 1st world and the 35,000th world was equal to -40.04 and had a standard deviation of 25.58. Running the paired t-test resulted in a t-value of -14.01 and an associated p-value much less than 0.001. Therefore, H_0 is rejected and we conclude that there is very strong evidence that the Q-learning model helped to increase the performance of the agents in a *SimpleWorld* environment.

Predators and obstacles were clearly avoided by the agent; however, food and poison was either missed or consumed respectively. This is due to the level of uncertainty in the agent's sensor readings for these two objects. Nevertheless, the experiment was successful in showing that the Q-learning algorithm could be applied to an agent within the *SimpleWorld* model to produce a reasonable controller.

4.4 The Desires Network/Action Module Experiment

The purpose of this experiment was to explore the effectiveness of the desires network and action module combination. Using a *SimpleWorld*, the agent was evolved to produce a set of desired sensor readings. It uses Q-learning during its lifetime to try to choose actions which will produce those desired sensor readings. The goal was to try to show that the combination of the two modules could produce an agent that would act sensibly in the world. The attractiveness of this simulation was its speed. Using the Mac Xgrid system, individuals in the population could be run in parallel and each individual would need only a few seconds to complete its run in 35,000 *SimpleWorld*s.

To set up this experiment, resources from the dLife package were used to run the genetic algorithm, construct the feed forward neural network for the desires network, and run the trials across the Xgrid.

Each individual in the population represented one agent in the *SimpleWorld*. Its genes represented the weights of the desires neural network. For this network, the number of inputs and outputs is four (one for each current sensor as input and one for each desired sensor as output). Additionally, a single hidden layer existed in the network that consisted of four neurons. At the beginning of the trial, all of the genes (i.e. weights) were randomized from a Gaussian distribution with a mean of 0 and standard deviation of 0.05 for all 100 individuals in the population.

After the population is created, it is set to run for 150 generations using a tournament selector with a tournament size of three. The mutation rate for the genes was 0.7 with a standard deviation of 0.05. This gives the genes a high probability of mutating in small amounts. The fitness of each individual would be the average lifetime of the agent over 35,000 *SimpleWorld* environments.

4.5 The Desires Network/Action Module Results

During this experiment, the minimum, average, and maximum fitness values (the agent's average lifespan over its 35,000 *SimpleWorld* environments) of each population were recorded. As with the Q-learning experiment, the results were relatively positive. The agent was able to evolve a correct desires network and learn over its lifetime what actions to take in order to achieve the desired sensor values. Figure 3 depicts the results of the average and best individuals of each population.

It is clear that the agents are able to use evolution to improve which sensory desires to look for and use Q-learning to take appropriate actions that eventually achieve these sensor values. This was also observed visually in the GUI.

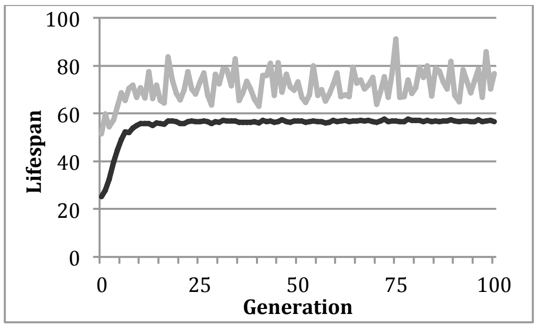

Figure 3: The average and best fitness values of each population over 100 generations.

5. Conclusion

The overall goal of both experiments was to verify that the combination of both the desires network and action module could be used to successfully evolve an agent that acts sensibly. From the results in the previous section, it is clear that this approach can be used in practice to evolve such an agent.

While the average lifetime of the agents in the final experiment was smaller than those of an experiment that only relied on Q-learning, several factors could contribute to this discrepancy. When thinking critically about what exactly the agent in the *SimpleWorld* was trying to predict, some information seemed to be missing. For example, when an agent is three spaces away from a food object, the sensors are green and sweet smelling. Now, if the agent takes a step forward, its sensors remain unchanged; however, once the food is consumed, the sensor values could be anything. Therefore, information about the agent's location relative to food seems to be missing from its state. This appears to be a limitation of the sensor system in the *SimpleWorld* rather than the approach. As stated earlier, these results demonstrate that this framework could indeed provide sensible results when attempting to bridge the reality gap.

6. Future Investigations

The *SimpleWorld* experiment has been a key component in testing the effectiveness of the proposed reality-gap bridge. The current goal is to work on the reality gap problem. During the continuing research, the Khepera III mobile robot will be used. While this robot has many unique features, the experiment will rely solely on its infrared proximity sensors and its differential drive system. The Khepera III is equipped with nine outward facing infrared proximity sensors that return an integer value from 0 to 4096. Higher values indicate a more intense infrared light and thus a closer object. This information is useful for wall following or obstacle avoidance. The robot is also equipped with two DC motors that can control the speed of both the left and right wheels independently.

In order to conduct experiments relating to the reality gap, the Khepera III will need to be run in a simulated environment. The simulation software that will be used is the Player/Stage robotics simulation environment [13]. With work from the past academic year, the use of the dLife interface into Player/Stage can be utilized. This allows the modeling of a Khepera III within a simulated world and allows faster-than-real-time experiments to run. Additionally, simulations will be allowed to take place across the Xgrid system to allow trials to take place in parallel.

6.1 The Reality Gap Experiment Design

Similar to the desires network and action modules in the *SimpleWorld* experiment, populations of robot controllers will be evolved; however, their goal this time is to achieve basic obstacle avoidance. While the problem of obstacle avoidance is not necessarily a hard problem to solve, the goal of the research is to examine the proposed reality-gap technique. Therefore, a harder problem should follow the same principles. Each individual will be evaluated using a standard fitness function that has been used in multiple obstacle avoidance simulations across the field [14]. The fitness function is shown in equation 3.

$$F = \sum_{i=1}^{k} \frac{R_{l_i}+R_{r_i}}{2}\left(1 - {V_i}^2\right)\left(1 - \max(P_i)\right) \qquad (3)$$

In equation 3, F represents the total fitness for the trial, k is the number of sensor readings which take place during our time interval, R_l is the rotational speed of the left wheel, R_r is the rotational speed of the right wheel (both scaled from -1 to 1), V_i is the difference in the left and right wheel speeds (scaled from -1 to 1), and P_i is the current maximum sensor reading of all its proximity sensors (scaled from 0 to 1). With this fitness function, robots will be given higher fitness for having low sensor readings over time combined with a reward for moving around (in particular moving in a straight line). This also prevents the robot from adopting known trivial solutions such as sitting still or turning in a circle the entire time.

Similar to the *SimpleWorld* experiment, this robot controller will use the desires network and action module framework to both learn throughout its life and choose its actions. The reinforcement signal will be 1 over the Euclidian distance between the desired sensor values and the actual sensor values as shown in equation 4. Here, R represents the reinforcement signal, $P_{i,d}$ and $P_{i,a}$ are equal to the desired and actual sensor values respectively, and n is equal to the number of sensors.

$$R = 1 \Big/ \sum_{i=1}^{n}\left(P_{i,d} - P_{i,a}\right)^2 \qquad (4)$$

As with the *SimpleWorld* experiment, the agent will be given no information during its lifetime on if these desires actually are correct. The desires network will only learn to perform actions that result in such desired sensors. It is the job of evolution to produce a robot that desires correct sensor values that achieve obstacle avoidance.

After each generation, the data is analyzed and a selection is made based on fitness to produce the next generation of robot controllers and the process continues. At the end of the experiment, the controller with the highest fitness will be evaluated on the physical Khepera III robot. The goal is to have the controller first run in simulation that will begin to train the actions module. The controller will then be moved onto the physical robot and evaluated based on its ability to perform obstacle avoidance in the real world. During this time, it will continue to adjust its action module. It is at that point, that a valid evaluation of if the proposed solution does indeed help to bridge the reality-gap.

7. Acknowledgements

I would like to thank Prof. Grant Braught for his guidance, expertise on the subject matter, and use of the dLife software package.

References:

[1] D. Beasley, D. Bull, & R. Martin. An Overview of Genetic Algorithms: Part 1: Fundamentals, *University Computing*, *15*(2), 1993, 58-69.

[2] J-A. Meyer, P. Husbands, & I. Harvey. Evolutionary Robotics: A Survey of Applications and Problems. *Eighth European Workshop on Learning Robots,* 1998, 14-22.

[3] N. Jakobi. Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis, *Adaptive Behavior*, *6*(2), 1997, 325-368.

[4] C. Hartland, & N. Bredeche. Evolutionary Robotics, Anticipation and the Reality Gap. *ROBIO 2006*, Kunming, China, 2006, 1640-1645.

[5] S. Koos, J-B. Mouret, & S. Doncieux. Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers. *Genetic and Evolutionary Computation Conference*, New York, NY, 2010, 119-126.

[6] B. Coppin. *Artificial intelligence illuminated* (Sudbury, MA: Jones and Bartlett, 2004).

[7] D. Whitley, S. Dominic, R. Das, & C.W. Anderson. Genetic Reinforcement Learning for Neurocontrol Problems, *Machine Learning*, *13*(2), 1993, 259-284.

[8] L.P. Kaelblin, M.L. Littman, & A.W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research, 4,* 1996, 237-285.

[9] R.S. Sutton, & A.G. Barto. *Reinforcement learning: an introduction* (Cambridge, MA: MIT Press, 1998).

[10] S. Russell, & P. Norvig. *Artificial intelligence: a modern approach* (Upper Saddle River, NJ: Prentice Hall, 2010).

[11] L. Lin. Self-Improving Reactive Agents Based On Reinforcement Learning Planning and Teaching, *Machine Learning*, *8*, 1992, 293-321.

[12] G. Braught. dLife – Robotics, AI, and Vision in Java, 2011. Accessed 3/01/2011: http://users.dickinson.edu/~braught/dlife/dLife/dLife.html

[13] Player Project, 2010. Accessed 3/01/2011: http://playerstage.sourceforge.net/

[14] O. Miglino, H.H. Lund, & S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments, *Artificial Life*, *2*, 1996, 417-434.