Evolving Robotic Desires: A New Approach to Bridging the Reality Gap

by

Russell Toris

Submitted in partial fulfillment of Honors Requirements
for the Computer Science Major
Dickinson College, 2011

Prof. Grant Braught, Supervisor
Prof. John MacCormick, Reader
Prof. Becky Wells, Reader

May 17, 2011

The Department of Mathematics and Computer Science at Dickinson College hereby accepts this senior honors thesis by Russell Toris, and awards departmental honors in Computer Science.

Grant Braught (Advisor)	Date
John MacCormick (Committee Member)	Date
Becky Wells (Committee Member)	Date
Tim Wahls (Department Chair)	Date

Department of Mathematics and Computer Science
Dickinson College

May 2011

Abstract

Evolving Robotic Desires: A New Approach to Bridging the Reality Gap

by
Russell Toris

Evolutionary Robotics is an expanding area in the world of robotics that incorporates ideas from fields such as Biology, Engineering, and Computer Science. The main idea behind Evolutionary Robotics is to use computerized models of biological evolutionary phenomena to *evolve* robotic behaviors. In recent years, new breakthroughs have been made to further its development, prove its effectiveness, and provide solutions to interesting problems within the robotics world. The approach of evolving controllers for autonomous robots has established benefits over 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 its performance tends to decrease on a given task; this issue is referred to as the reality gap problem. In this research, a new approach to bridging the reality gap is presented and explored. The idea is to evolve a robotic controller that generates desires based on its current state and uses reinforcement learning to select actions that achieve these desires. By doing so, the goal is to have a robotic controller adapt to differences, uncertainties, and perturbations within the real world once transferred from simulation.

Acknowledgments

I would like to thank Prof. Grant Braught for his guidance, expertise on the subject matter, and support for the dLife software package. Furthermore, I would like to thank Prof. John MacCormick and Prof. Becky Wells for their feedback throughout the year, as well as Prof. Dick Forrester for his advice on the statistical tests.

Table of Contents

Title Page i
Signature Page ii
Abstract. iii
Acknowledgements iv
Table of Contents. v

Chapter 1: BACKGROUND. 1
1.1. Evolutionary Robotics 1
1.2. The Reality Gap Problem. 2
1.3. Related Work. 3

Chapter 2: A NEW MODEL TO BRIDGE THE REALITY GAP. 6
2.1. The Desires Network and Action Module 6
2.2. The Desires Network. 7
2.3. The Action Module. 10
2.3.1. Q-Learning 11
2.4. The Complete Framework. 14

Chapter 3: EXPERIMENTAL PROOF OF CONCEPT 15
3.1. Simplifying the Problem 15
3.1.1. The *SimpleWorld* Model. 15
3.2. The *SimpleWorld* Q-Learning Experiment. 17
3.2.1. The Q-Learning Results 19
3.3. The Desires Network/Action Module Experiments 20
3.3.1. The Current Sensor and Desires Action Selection Results 23
3.3.2. The Desires Only Action Selection Results 24
3.4. An Analysis of the *SimpleWorld*. 25

Chapter 4: ROBOTIC APPLICATION OF THE PROPOSED MODEL. 30
4.1. The Khepera III 30
4.2. The Simulated Robot Experiment 32
4.3. Discretizing States and Actions. 34
4.3.1. The Simulated Robot Results. 35
4.3.2. Transfer of the Controller to the Real Robot 37
4.4. The Descriptive Desires. 39
4.4.1. The Descriptive Desires Results 40
4.4.2. Transfer of the Descriptive Controller to the Real Robot 43

Chapter 5: CONCLUSION 44
5.1. The Framework and *SimpleWorld* Experiments. 44
5.2. The Simulated Robot Experiments 45
5.3. The Reality Gap. 47

Chapter 6: FUTURE WORK AND EXTENSIONS. 49
6.1. Future Work. 49
6.2. Extensions. 51

References 52

Chapter 1

BACKGROUND

1.1. Evolutionary Robotics

The area of Evolutionary Robotics is a relatively new research field that incorporates ideas from fields such as Computer Science, Engineering, and Biology. The general approach is to use a genetic algorithm to *evolve* sensible robotic controllers. These algorithms work on the basic theory of biological evolution and attempt to make use of natural evolutionary phenomena such as survival of the fittest (Beasley, Bull, & Martin, 1993). The idea is to use computational models of these phenomena to produce a solution to a complex problem within some domain.

In Evolutionary Robotics, a genetic algorithm is used with a population of hundreds of individuals that represent random robot controllers (a software program used to govern the actions of a robot). These controllers are then encoded with a fixed number of *genes* that represent properties of the controller. The next step is to evaluate the performance (also known as the fitness) of each controller and reproduce a selection of the best performing ones over a few hundred generations. The reproduction method will take the genes of selected individuals from the population and create a new population of individuals. Different reproduction and selection methods can be used such as crossover (sexual reproduction), mutation of the genes by a fixed probability, and tournament selection (Beasley, Bull, & Martin, 1993). A basic outline of a genetic algorithm is

seen in Algorithm 1.1 and returns the best individual (i.e. the best performing robot controller).

Algorithm 1.1: GeneticAlgorithm

```
# Pre: Gens is the predefined number of generations to run
       evaluate will evaluate and store the fitness for the Individual

Individual GeneticAlgorithm {
        Pop ← new population of random robot controllers
        CurGen ← 0
        while (CurGen < Gens) {
                for each Individual in Pop {
                        evaluate(Individual)
                }
                Pop ← selection/mutation of current Pop
                CurGen ← CurGen + 1
        }
        return Individual in Pop with highest fitness
}
```

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 absence 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 and have been able to overcome changing environments or the inability to anticipate every possible situation (Meyer, Husbands, & Harvey 2008).

1.2. The Reality Gap Problem

Researchers in Evolutionary Robotics have encountered many 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 where each generation typically has a population of hundreds of

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. It is impractical or financially impossible to solve this problem by using multiple robots. To overcome this, researchers have turned to the use of simulators to conduct their trials. Simulators provide the means to run trials at a faster rate and also allow multiple trials to be run in parallel.

This may sound like an 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 take advantage of properties and certainties that exist in the simulator but not in the real world. Thus, even the fittest controllers from simulation can fail when run on robots in the real world. Many researchers refer to the decrease in performance when a controller is transferred from simulation to the real world as the reality gap problem (Jakobi, 1997; Hartland & Bredèche, 2006; Koos, Mouret, & Doncieux, 2010). The research presented here will focus on examining and testing a novel method for closing this gap.

1.3. Related Work

Several approaches have been proposed to bridge the reality gap. One such method is the radical envelope-of-noise (Jakobi, 1997). This method attempts to

place a reasonably high amount of noise onto the simulation sensors while running the evolution trials. The idea is that the robot will not become dependent on artifacts of the simulation because they are masked by the noise. In addition, any robot controller that performs with high fitness under such conditions should have the ability to deal with variability in the real world.

Another approach is to incorporate an anticipation model into the robot's controller (Hartland & Bredèche, 2006). 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. An estimation of the amount of error between simulation and the real world can be obtained by comparing its actions and their associated effects in simulation against those in the real world. This information can then provide correction criteria for use by the controller when determining what actions to make in the real world. The controllers use this error estimation and attempt to adjust themselves accordingly while running in the real world.

A third approach is to evaluate and promote “transferable controllers” (Koos, Mouret, & Doncieux, 2010). The idea proposed here is to run a traditional genetic algorithm experiment and evaluate fitness accordingly. An additional criterion is then used to look for novelty and diversity within the simulated controllers. 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. Controllers that perform similarly on physical robots and on their simulated counterparts are

promoted to the next generation. This method rewards controllers that have similar behaviors in the real world and in the simulation. As a result, this method has been shown to evolve highly transferable controllers.

Chapter 2

A NEW MODEL TO BRIDGE THE REALITY GAP

2.1. The Desires Network and Action Module

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 novel approach to bridging the differences between the simulated and real world environments. The goal is to have a controller that can adapt to changes and uncertainties by not using its sensors to directly determine its actions. Instead, the controller will use its sensors as a learning criterion that can be used to adapt its actions as is necessary.

Each robot controller will use two separate modules to determine its actions. The idea is to incorporate an evolved neural network that produces a set of *desires* based on the robot's current state (e.g. sensory values). For example, if a robot that is attempting to perform obstacle avoidance is too close to an obstacle, it may desire to be further way. The desires produced by the network and the current state information will then be fed into another module that will use reinforcement learning to find actions that satisfy the desires. These actions and desires can be defined based on the specific problem at hand; in the previous example, one such action might be to turn away from the obstacle. The two modules that will be used will be referred to as the desires network and the action module. An abstract visualization of these modules is depicted in Figure 2.1.

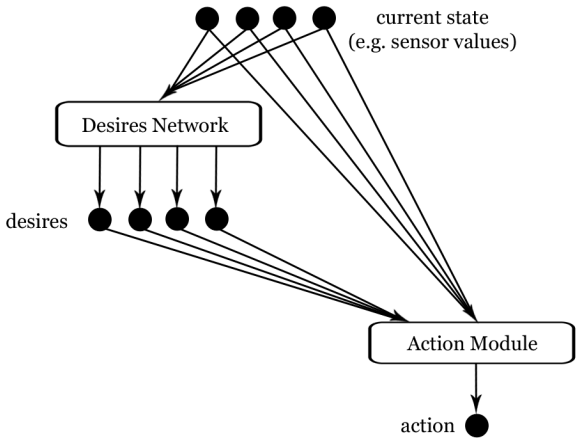

Figure 2.1: The basic outline and integration of the desires network and action module.

The hope is that this framework will evolve an agent that does not create a direct relationship between its current state and an action. Rather, the desires network will pre-process the state information and then rely on the action module to select appropriate actions. Thus, if the controller evolves to rely less on its current state information directly, it may be able to adapt to the differences between the real world and simulated environments.

2.2. The Desires Network

The desires network will use a feed forward neural network (Coppin, 2004). A neural network can be thought of as a weighted directed graph where each node in the graph is referred to as a neuron. The initial layer of neurons is the input layer and the final layer of neurons is the output layer. Additionally, a layer of neurons can be placed between the input and output layers. This layer, referred to as the hidden

layer, contains an arbitrary number of neurons. A generic model of a neural network is shown in Figure 2.2.

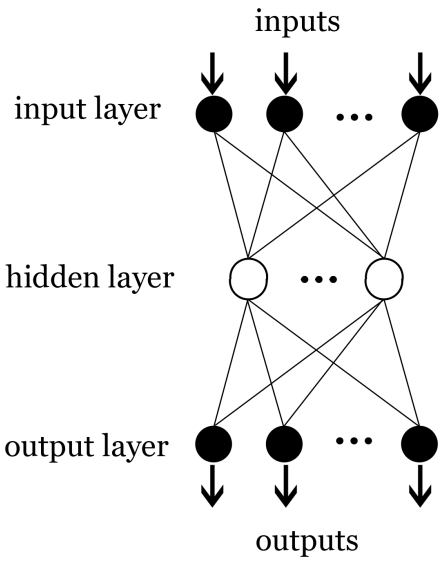

Figure 2.2: A generic visualization of a feed forward neural network.

The basic process of the neural network is to take the values of the inputs, pass them through each layer of neurons, and produce a set of output values. To begin, the network takes each input value and passes it into a corresponding neuron on the input layer. Each neuron has an activation function that can be used to compute the final output of the neuron based on its input. For example, a sigmoid function, shown in Equation 2.1, can be used to bind continuous values to be within the range of 0 to 1.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

A filter can also be used in conjunction with this activation function to shift and scale the range of possible values. Furthermore, other activation functions can also be used depending on the application of the network (Coppin, 2004).

As values are pushed through the network, the weighted edges connecting them are used to scale the values. The net input to each neuron is computed as shown in Equation 2.2 where w_i and v_i are the weight and value of the current input respectively, and n is the total number of inputs to the neuron.

$$X = \sum_{i=1}^{n} w_i v_i \tag{2.2}$$

A more detailed visualization of the neuron can be seen in Figure 2.3. This illustration shows that in addition to the weights from each neuron, a bias value and weight are assigned to each neuron in the hidden and output layers with a constant input value of 1. This bias is necessary for shifting and transforming the activation function. Furthermore, it should be clear that as the weights of the feed forward network change, the outputs would change given the same inputs.

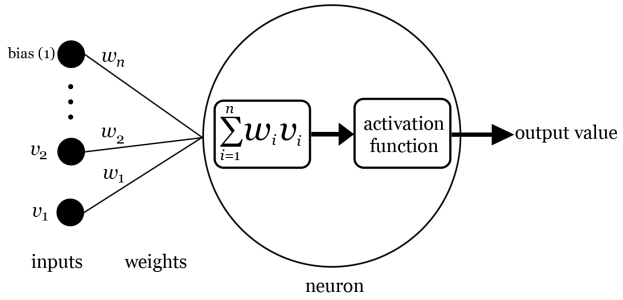

Figure 2.3: A detailed visualization of a single neuron.

The purpose of this desires neural network is to have the controller express properties about what its next state should look like based on the current state. For the purposes of the desires network, the input to the neural network is the controller's current state information gathered from the robot and has the same number of outputs as inputs. The inputs represent each of the associated values of the state and the outputs represent the *desires* of the controller. The network will

also contain a single hidden layer with additional neurons. The weights of the 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.

2.3. The Action Module

The current state information and the desires generated by the desires network are then fed into the action module that produces a single action that will be executed. 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) based on the action it selects, and have the module adjust itself accordingly (Whitley, Dominic, Das, & Anderson, 1993; Kaelblin, Littman, & Moore, 1996; Sutton & Barto, 1998; Russell & Norvig; 2010). The module will be given positive feedback when it causes an action that produces a new state that satisfies its desires and vice versa. The proposed feedback (referred to as a reinforcement signal) is defined to be a distance measurement between the resulting state and the previous desires. Depending on the application, this measurement will have different implementations.

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, which are produced by its desires network. Depending on the desires network, the actions produced from 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 and will therefore perform poorly on the evolutionary task. Thus, the fitness function defined within the experiment will assign relatively low fitness to these individuals and will inhibit them from reproducing. Over time controllers will evolve to produce desires that are appropriate to solve the problem and the action module will learn to take actions that satisfy those desires.

2.3.1. Q-Learning

While the general approach is to use reinforcement learning, there are several different variations. The proposed type of reinforcement learning for the action module is Q-learning (Lin, 1992; Russell & Norvig, 2010). This approach attempts to determine and provide outputs that will maximize the reinforcement signal (called the Q-function).

Q-learning works by finding and executing the *best action* across all possible actions for the current *state*. Within the action module, the term state can be any combination of the *desires* and the controller's current *state* information. Therefore, when referring to the state within this module, it will be referring to one of the following depending on the experiment: just the agent's current state (i.e. sensor) information, just the agent's desires information, or both the agent's current state and desires information. Note that even if the action module may not use the agent's current state information during action selection, it will still need to be provided this information to compute the reinforcement signal.

In Q-learning, each action is ranked with a value referred to as the Q-value. This value is updated over time to represent how well a given action increases the reinforcement signal from the current state (referred to as the state-action pairing). The action chosen is also contingent on 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 pairing has been observed and return a single value. The action that maximizes the exploration function is chosen. For this research, the exploration function seen in Equation 2.3 was implemented. This was an adaptation of an example presented by Russell and Norvig (2010; p. 842).

$$f(q,n) = \begin{cases} R, & n < N \\ q, & \text{otherwise} \end{cases} \tag{2.3}$$

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 during a given experiment, R is the maximum reward possible in any state, and N is a specified 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.

Q-learning keeps track of the Q-values for all possible state-action pairs in a structure called the Q-table. The algorithm updates these values during each request for an action using the function defined in Algorithm 2.1 that is adapted from Russell and Norvig (2010, p. 844). The algorithm takes a *percept* that contains the information about the current state and the current reward value from the reinforcement signal. In addition to the Q-table, a frequency table (referred to as the

N-table) stores how many times a given state-action pairing has been observed. When an iteration of the algorithm is run, both the Q-table and N-table are updated.

In addition to this information, the update function for the Q-values requires both an alpha (α) function and gamma (γ) value. The alpha function is used in order to help the Q-table converge and is implemented as an "alpha step function" (Russell & Norvig, 2010). The alpha step function used is shown in Equation 2.4 where λ represents a set value (which will be referred to as the alpha step size value). This function acts as the learning rate for the Q-value update function and is used to determine how much to value newly observed rewards over time.

$$\alpha(n) = \frac{\lambda}{\lambda + n - 1} \tag{2.4}$$

The value of gamma (γ) is the discount factor and is a set value between 0 and 1. This value is used to determine how much to consider future Q-values (i.e. the future reward it thinks it will receive for taking the action in the current state).

Algorithm 2.1: Q-Learning

```
# Pre: QTbl is the Q-table/N-table indexed by actions and states
       NTbl is the N-table indexed by actions and states
       Percept contains the current state (s) and reward (r)
       s', a', and r' are the previous state, action, and rewards
       α the alpha step function
       γ the discount factor (between 0 and 1)
       f is the exploration function

Action Q − Learning(Percept) {
        if (s' ≠ null) {
                NTbl[s', a'] ← NTbl[s', a'] + 1
                QTbl[s', a'] ← QTbl[s', a'] + α(NTbl[s', a']) * (r' +
                                γ max_a QTbl[s, a] − QTbl[s', a'])
        }
        s' ← s,
        a' ← argmax_a f(QTbl[s, a], NTbl[s, a])
        r' ← r

        return a'
}
```

2.4. The Complete Framework

In the proposed framework, each controller will have an implementation of both the desires network and the action module. 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 learned information will not be passed on to future generations. As with natural evolution, what one learns in life is not directly passed on genetically. At the beginning of each trial, the controller will be given a standard default action module where all the values in the Q-table will be initialized to 1, all the values in the N-table will be initialized to 0, and the previous action, state, and rewards will all be *null*. Throughout its lifetime, the agent will modify its own action module using Q-learning.

Chapter 3

EXPERIMENTAL PROOF OF CONCEPT

3.1. Simplifying the Problem

The first major task in the research is to investigate both a proof of concept and the effects of the information provided to the action module (i.e. just the controller's current sensors, just the desires, or both the desires and the controller's current sensors). This important step is necessary to verify components of the experiment in small increments before undertaking 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, etc.), it is unrealistic to go straight to the large-scale experiments. Therefore, the decision was made to test out the concept of using the desires network and action module on an agent in the *SimpleWorld*.

3.1.1. The SimpleWorld Model

The *SimpleWorld* is a discrete grid-world model that is part of the dLife software package (Braught, 2011). The world is a two-dimensional grid (such as the example depicted in Figure 3.1) with random assignments of food, poison, obstacles, and predators. The world will always contain ten predators, ten obstacles, ten poison objects, and fifteen food objects. Once a piece of food or poison is consumed, another will be randomly placed in the world. A single agent is then placed in the

center of the grid and has three possible actions that it can take: turn left, turn right, and move forward.

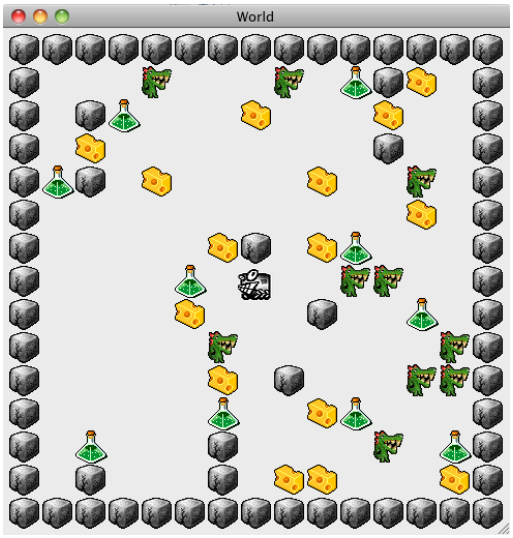

Figure 3.1: An example of a *SimpleWorld*.

The agent is given an initial amount of health, or life force, of 100. The life force 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. These values are given in Table 3.1.

Table 3.1: The life force penalties/rewards for a *SimpleWorld* agent.

Activity	Penalty/Reward
Food Eaten	+10
Poison Eaten	-5
Obstacle Collision	-2
Predator Collision	-10
Any Other Move	-1

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 3.2.

Table 3.2: 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. For the purposes of this research, a smell range of one and a color range of three were used. 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 the closest smell within a specified radius of itself. If more than one closest smell is detected, one will be picked at random. An important note is that both poison and food have a color value of *green*; however, they have different smell values. Because the smells are random, if they are equidistant from the agent this creates some 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.

3.2. The *SimpleWorld* Q-Learning Experiment

Three separate experiments were used as a proof of concept. The first experiment looked at having a Q-learning module that only used the agent's current sensor state information. Since no desires would be used in this module, there was

no need to incorporate a desires network, and thus, no use of evolution was required. This experiment evaluated the use of only machine learning to create a sensible agent in the *SimpleWorld*. This module would take in the agent's current state information (the sensor values) and produce one of the agent's actions.

For this experiment a cutoff of 7 was used for the evaluation function (the value of N in Equation 2.4). This forces the agent to try each of the possible actions at least seven times. This also helps to create a reasonable sample size for the Q-values before using them to decide on an action. Furthermore, a value of 0.6 was used for gamma (γ) and a value of 900 was used as the alpha step size value (λ). These values were determined to work reasonably well based on several initial experiments.

The reward for an action (the reinforcement signal) was 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. Exact reward values are those seen in Table 3.1.

As stated earlier, the goal of the experiment was to evaluate how well the action module could create a sensible agent (i.e. live as long as it can) in a *SimpleWorld* model. The experiment ran a single agent through 35,000 randomized *SimpleWorld* instances where a new instance was created after the agent died. The Q-table was not reset after each world; therefore, the agent would be modifying the Q-table and learning throughout the 35,000 worlds. 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 reiterate that no evolution was used in this experiment; the same agent is being run through thousands of *SimpleWorld* environments.

3.2.1. The 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. This could be seen both visually by observing the agent in a *SimpleWorld* as well as analyzing the data. Table 3.3 summarizes the average lifespan of agents in the 1st world and 35,000th world across the 100 trials of the experiment.

Table 3.3: The lifespan of an agent in the 1st and 35,000th world averaged across 100 trials.

World	Average Lifespan	Standard Deviation
1st	25.81	11.24
35,000th	61.10	23.82

In order to compare the means from the trials, a paired *t*-test was performed to test if the Q-learning model helped to improve the agent's performance in the *SimpleWorld* (Walpole, Myers, Myers, & Ye, 2006). The appropriate null and alternative hypotheses are shown in Equation 3.1 where μ_D is the mean of the differences between lifespans from the 1st world and the 35,000th world.

$$H_0: \mu_D = 0$$
$$H_1: \mu_D < 0 \qquad (3.1)$$

The mean of the differences between lifespans from the 1st world and the 35,000th world was equal to -30.91 and had a standard deviation of 26.41. Running

the paired *t*-test resulted in a *t*-value of -13.36 and an associated *p*-value much less than 0.001. Therefore, H_0 is rejected and it is concluded that there is very strong evidence that the Q-learning model helped to increase the performance of the agents in a *SimpleWorld* environment.

Qualitatively the agent was observed avoiding predators and obstacles; however, food and poison was occasionally missed or incorrectly consumed respectively. The occurrence of incorrectly consuming poison is due to the non-determinism in the agent's smell sensor readings between food and poison. For example, if the agent has a food object on its left and a poison object on its right, both its left and right color sensors would be green and its smell sensor would have a 50% chance of being either sweet or sour. Therefore, an agent could not learn to overcome this uncertainty based on the available information. This 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.

3.3. The Desires Network/Action Module Experiments

The final two experiments were to test the desires network and action module pairing in an evolutionary experiment. In the first experiment, the action module would use both the agent's current sensor state information and its desires when selecting actions. In the second experiment, the action module would only use the agent's desires (produced by the desires network) when selecting actions. The effectiveness of these combinations and the Q-learning experiment is then explored.

Using a *SimpleWorld*, the agent evolved the weights of the desires neural network to produce a set of desired sensor readings given its current sensor readings (i.e. the agent's current state). It uses Q-learning in its action module during its lifetime to try and 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 a sensible agent that was at least equivalent to the Q-learning only agent. 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.

Each individual in the population represented one agent in the *SimpleWorld* and 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 three neurons. Since the agent's sensors were based on colors or smell, numerical values were assigned to each possible value as seen in Table 3.4.

Table 3.4: The numerical sensor values for the color and smell sensors.

Smell	
sweet	1
sour	-1
none	0

Color	
green	1
red	-1
none	0

Since a sigmoid activation function was used (Equation 2.1), a filter was needed to map the raw output of the neuron to one of the three possible sensor

values. Therefore, Equation 3.2 was used where F returns the filtered output and r represents the raw output of the neuron.

$$F(r) = \begin{cases} 1 & r \geq 0.66 \\ 0 & 0.66 > r > 0.33 \\ -1 & r \leq 0.33 \end{cases} \quad (3.2)$$

The agent's current state and the resulting desires are then fed into the action module. In the first experiment, both of these separate pieces of information are combined as one state. Therefore, the size of the state space in the action module is 3^8 or 6,561. In the second experiment, only the desires information was used as the action module's state. Therefore, the size of the state space in the action module is 3^4 or 81. While the size of the state space in the first experiment is much larger than that of the second experiment, each agent is being run through 35,000 *SimpleWorld* instances and takes around 25-80 actions in each. Thus, the agents in both experiments will have ample time to explore the majority of the state space.

At the beginning of the trial, all of the genes (i.e. weights) for all 100 individuals in the population were randomized from a Gaussian distribution with a mean of 0 and standard deviation of 0.05. After the population was created, it was set to run for 125 generations using a tournament selector with a tournament size of three. This selection model chooses three individuals at random and selects the one with the highest fitness for reproduction (Beasley, Bull, & Martin, 1993). This process is then repeated 100 times until a new population is created. The mutation rate for the genes was 0.75 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.

As with the Q-learning experiment, a smell range of one and color range of three was used. Furthermore, each individual is run across 35,000 randomized worlds attempting to learn what actions to make using the desired and current sensor readings. Again, a cutoff of 7 was used for the exploration function, a value of 0.6 was used for gamma (γ) and a value of 900 was used as the alpha step size value (Y). The reward used was the reciprocal of Euclidean distance between the desired sensor readings and the resulting sensor readings as shown in Equation 3.3. 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 total number of sensors (in the case of the *SimpleWorld*, n will be equal to 4). If the Euclidean distance was equal to 0 (i.e. it matched the desired state exactly) a reward of 1 was returned. Note that 1 is the maximum value this function could ever return. This is due to the fact that $P_{i,d}$ and $P_{i,a}$ are integer values between -1 and 1 (inclusive). Thus, the sum of the squared differences (the denominator) must be either 0 or a positive integer value.

$$R = \begin{cases} \frac{1}{\sum_{i=1}^{n}\left(P_{i,d}-P_{i,a}\right)^2} & \text{if} \sum_{i=1}^{n}\left(P_{i,d}-P_{i,a}\right)^2 > 0 \\ 1 & \text{otherwise} \end{cases} \quad (3.3)$$

3.3.1. The Current Sensor and Desires Action Selection Results

During this experiment, the minimum, average, and maximum fitness values (the agent's average lifespan over its 35,000 *SimpleWorld* environments) of the population were recorded for each generation. As with the Q-learning experiment, the results were positive. The agent was able to correctly evolve a desires network that produced sensible values for the action module to use as learning criteria when

selecting actions. Figure 3.2 depicts the results of the average and best individuals in the population over the 125 generations of the experiment.

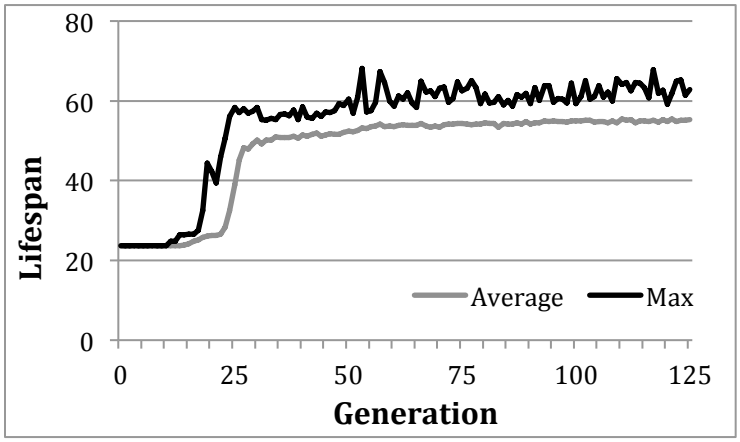

Figure 3.2: The average and best fitness values of the population over 125 generations.

It is clear by the rise in fitness that the agents are able to use evolution to improve the controller's desires and use the Q-learning model in order to take appropriate actions that satisfy those desires. This was also observed visually in the GUI. One important feature of this graph is that the agents did slightly worse than the previous experiment, which used only Q-learning. This is true even for many of the best agents in the population. A more detailed analysis of these results is presented in Section 3.4.

3.3.2. The Desires Only Action Selection Results

This next experiment used only the agent's desires when selecting actions. As with the previous section, the minimum, average, and maximum fitness values of each generation were recorded. The results here were extremely positive. The average fitness of the population was able to exceed the averages from both of the

previous experiments. Figure 3.3 depicts the results of the average and best individuals in the population over the 125 generations of the experiment.

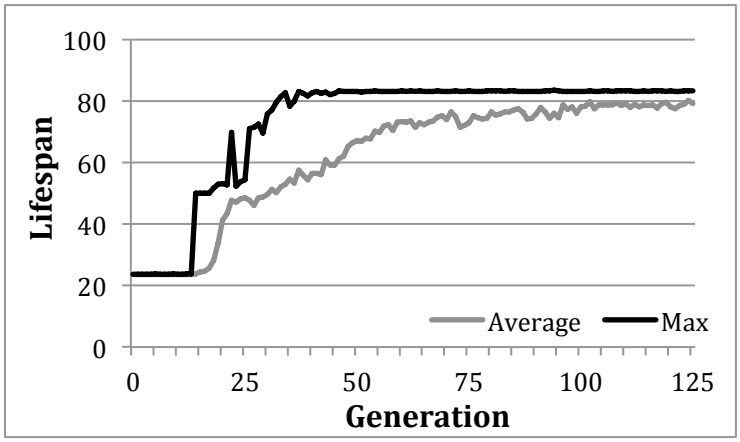

Figure 3.3: The average and best fitness values of the population over 125 generations.

3.4. An Analysis of the *SimpleWorld*

The overall goal of these experiments was to analyze and evaluate the performance of the desires network and action module for successfully evolving a *SimpleWorld* agent that acts sensibly. From the preliminary results observed in the previous two sections, it is clear that this approach can be used in practice to evolve such an agent.

The next step was to analyze which of the above methods could produce the *best* agent (if any) and theorize which properties of this method were able to give the agent an advantage. To test which method produced the best agent a small-scale experiment was run.

In this experiment, ten agents from each of the above experiments were selected. For the Q-learning experiment, each initial agent was identical; therefore,

no selection of a specific individual was needed. For the desires network/action module experiments, the best agent of the population in each of the final ten generations was selected. With the agents selected from each method, each was trained individually through 35,000 random *SimpleWorld* instances and their average lifespan was recorded. This process was repeated ten times for each individual. Thus, each method contained 100 pieces of data. The results are displayed in Table 3.5 and Figure 3.4.

Table 3.5: The average lifespan of 10 agents of each method in 35,000 worlds averaged across 10 trials.

Method	Average Lifespan	Std. Dev.
Sensors Only	63.740	4.546
Desires Only	82.932	0.144
Both Desires & Sensors	55.545	2.696

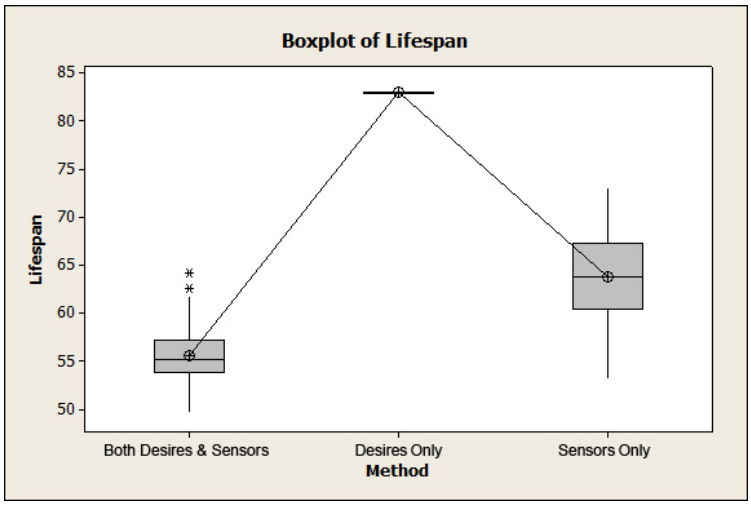

Figure 3.4: Boxplot of the average lifespan for the three different methods.

By looking at the data presented above, it appears that the agents that used only the desires information performed the best. A one-way ANOVA was performed

to analyze if there are differences in the mean lifespan of the agents between the three methods (Walpole, Myers, Myers, & Ye, 2006). The results of this test are presented in Table 3.6.

Table 3.6: ANOVA results for the three methods.

Source	Degrees of Freedom	Sum of Squares	Mean Square	F	p
Method	2	39517.25	19758.63	2120.40	<< 0.001
Error	297	2767.55	9.32		
Total	299	42284.80			

With such a small p-value produced by this test, it is clear that at least one of the methods produced an agent with a mean lifespan that was significantly different than the rest. While the ANOVA's assumption of constant variance was clearly violated, its normality assumption did hold. Furthermore, it is clear from the boxplots seen in Figure 3.4 that the experiment that only used the desires when choosing actions was the best.

When thinking about the difference between the desires-only representation and the combination of the desires and current sensor state, several factors could contribute to the improvement for the desires-only representation. One factor is the fact that the state space for the action module is smaller for the desires-only state representation. This state has four desired sensors each with three possible values giving a state space of 81. The combination of both the current sensors and desired sensors has four desired sensors and four current sensors each with three possible values giving a state space of 6,561. While these numbers may seem significantly different, it is not responsible for the difference in performance.

To understand why, it is important to notice that since the desires network is static for a given agent during its lifetime, it will always generate the same desires for a given state. Thus, the state space any given agent will see in their lifetime is reduced to 81. In other words, the state is determined entirely by the current sensor values rather than the produced desires. In a sense, the current sensors are masking the information about the desired state when they are sent into the action module. Since the combination of the desires and current sensor state experiment did worse than the experiment that only used Q-learning, it is suggested that the reinforcement signal used in the evolutionary experiment was preventing the agent from learning better. The experiment that ran without desires got direct feedback from the *SimpleWorld* for its reinforcement signal and was able to learn more effectively.

What is interesting and significant to look at is the difference in performance between the desires-only representation and the current-sensor-state-only representation. Both methods had four sensor values being sent into the Q-learning algorithm with the same possible set of values. Furthermore, as discussed earlier, the reinforcement signal used in the current-sensor-state-only experiment was providing direct feedback from the *SimpleWorld* on the agent's performance. While research has been done to show that evolution can be used to produce training information for supervised learning models (Nolfi & Parisi, 1993), these results suggest that evolution is also capable of producing values that are useful for reinforcement learning. Evolution is able to create a network to pre-process the

sensor information and reorganize the inputs to the action module with values (i.e. desires) that are easier to learn than the original sensor values.

These results show that this framework is capable of producing better agents that perform more consistently in a *SimpleWorld*. A further analysis of this framework is discussed in Chapter 5. Furthermore, since it has been shown that ignoring the current-sensor state when choosing actions is beneficial, a desires-only representation will be used in the subsequent experiments.

Chapter 4

ROBOTIC APPLICATION OF THE PROPOSED MODEL

The *SimpleWorld* experiment has been a key component in testing the effectiveness of the proposed reality-gap bridge. The next step was to investigate the framework's performance in a robotics domain. This chapter introduces and discusses the physical robot, simulator, and methods used to test the framework.

4.1. The Khepera III

During the research, the Khepera III mobile robot was used (K-Team Mobile Robotics, 2010). While this robot, seen in Figure 4.1, has many features, the experiment will use its infrared proximity sensors and its differential drive system.

Figure 4.1: A Khepera III mobile robot viewed from the front and top with infrared locations sensors circled.

The Khepera III is equipped with nine outward facing infrared proximity sensors that return an integer value from 0 to 4096. Higher IR 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 control the speed of the left and right wheels independently. These motors can be set with values in the range of -1 to 1 where -1 represents a full backwards speed and 1 represents a fill forwards speed.

Note that to reduce the size of both the neural network and Q-table, only the front six IR sensors will be utilized in the experiment described here. By decreasing the size of the network and Q-table, it should take less time for the controller to explore which actions to take, thus requiring less time in simulation. Once parameters are tweaked and the controller can take more time in simulation, the number of sensors can be increased and retested.

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 (Player Project, 2010). An example of the simulated environment the robot will be run in is shown in Figure 4.2.

Due to work from the previous academic year, the dLife interface into Player/Stage can be utilized (Toris, 2010). This allows a simplified Khepera III to be controlled with the same software controller that will run on the actual Khepera III and thus decreases variability in the experiment. Additionally, simulations will be distributed across the Xgrid system to allow trials to take place in parallel.

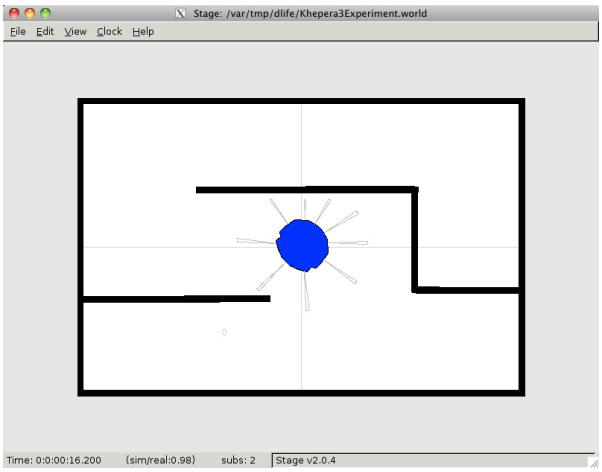

Figure 4.2: The simulated Khepera III in a Player/Stage simulated world.

4.2. The Simulated Robot Experiment

Similar to the *SimpleWorld* experiments, 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 hard to solve, the goal of the research is to examine the proposed reality-gap technique, not the problem of obstacle avoidance. Each individual will be evaluated using a standard fitness function that has shown to be effective for evolving obstacle avoidance (Nolfi, Floreano, Miglino, & Mondada, 1994). The fitness function is shown in Equation 4.1.

$$F = \sum_{i=1}^{k} \frac{R_{li}+R_{ri}}{2}\left(1-\sqrt{|V_i|}\right)(1-\max(P_i)) \qquad (4.1)$$

In Equation 4.1, F represents the total fitness for the trial, k is the total number of sensor readings which take place during the trial, 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, controllers will be given higher fitness for having smaller sensor readings and moving forwards quickly while turning as little as possible. This prevents the robot from adopting known trivial solutions such as sitting still or turning in a circle the entire time.

An individual in the population now represented one Khepera III controller. Similar to the *SimpleWorld* experiment, the controller used the desires network and action module framework both to learn throughout its life and to choose its actions. For the action module, a cutoff of 10 was used for the exploration, a value of 0.6 was used for gamma (γ) and a value of 900 was used as the alpha step size (λ). Again, the reinforcement signal was the reciprocal of the Euclidian distance between the desired sensor values (described in the next section) and the actual sensor values as shown in Equation 3.3 (where n is now equal to 6).

As with the *SimpleWorld* experiments, the genes of each individual represented the weights of the desires neural network. The number of inputs and outputs for the network was six (one for each sensor) and a single hidden layer existed in the network that consisted of four neurons. At the beginning of the trial, all of the genes were randomized from a Gaussian distribution with a mean of 0 and standard deviation of 0.05 for all 50 individuals in the population. The experiment was set to run for 200 generations using a tournament selector with a tournament size of three. The mutation rate for the genes was 0.75 with a standard deviation of 0.05. Furthermore, each individual was run in a noiseless simulated environment for 60 seconds.

As with the *SimpleWorld* experiment, the agent was not given information about whether or not its desires were correct during its lifetime. The action module blindly learned to perform actions that resulted in the desires being satisfied. It is the job of evolution to generate a controller with correct desires that achieves obstacle avoidance.

4.3. Discretizing the States and Actions

One important property of using the Q-learning in the action module is that the state and action spaces must be discrete and that states (i.e. the current desires) be uniquely mapped to an entry and return a single action; however, this is not the case for the state and action spaces of robot controllers. For example, the robot can control both its left and right motors by issuing them a value between -1 and 1, and each of the six sensor values can be any number between 0 and 4096. If the same model for desires were adapted from the *SimpleWorld* experiments (i.e. the desires represent what the sensor values should look like), the size of the state space would be 4096^6. This is clearly an impractical state space for the simulated robot to explore. Therefore, the decision was made to discretize both the action and state spaces. While this would limit the amount of information and freedom the robot had, the hope was that the basic concept and goal of the experiment could still be observed.

To split up the state space, cutoffs were used to assign one of three different values to each sensor: -1, 0, or 1. These bounded values can be thought of as representing that an obstacle is *very close, close,* or *far* respectively. Initially, the

continuous sensor values (scaled and shifted to the range -2 to 2) are fed into the desires network that would produce six different desires of either -1, 0, or 1 using the filter described in Equation 3.2. Therefore, the size of the state space was now 3^6 or 729. These numbers were then fed into the action module and used as the desired state. When calculating the reinforcement signal, the previous state (i.e. the sensor values) needed to be discretized as well. To accomplish this, Equation 4.2 was used where β takes the unbounded sensor value x and returns one of the three possible values.

$$\beta(x) = \begin{cases} -1 & x \geq 600 \\ 0 & 600 > x \geq 250 \\ 1 & x < 250 \end{cases} \tag{4.2}$$

To split the action space up, the controller was allowed to pick one action (or set of motor commands) from a discrete number of set motor commands. Table 4.1 shows the left and right motor commands for each of the 18 possible actions.

Table 4.1: The left and right motor commands for each of the possible actions.

	Fast	Medium	Slow
Hard Left	(-0.6, 0.6)	(-0.3, 0.3)	(-0.15, 0.15)
Slight Left	(0.0, 0.6)	(0.0, 0.3)	(0.0, 0.15)
Forward	(0.6, 0.6)	(0.3, 0.3)	(0.15, 0.15)
Slight Right	(0.6, 0.0)	(0.3, 0.0)	(0.15, 0.0)
Hard Right	(0.6, -0.6)	(0.3, -0.3)	(0.15, -0.15)
Backward	(-0.6, -0.6)	(-0.3, -0.3)	(-0.15, -0.15)

4.3.1. The Simulated Robot Results

Preliminary results from the initial robot experiment were positive. The data shows clear improvements in the best and average finesses of the population over

time. Figure 4.3 shows the average and best fitness values of the population for each generation of the experiment.

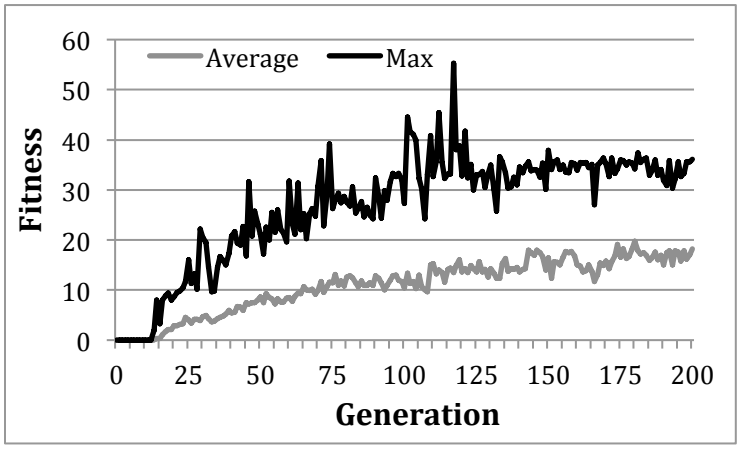

Figure 4.3: The average and best fitness values of the population over 200 generations.

After looking at the data, randomly selected best controllers from the final 25 generations were observed in Player/Stage. An example of the robot's path is depicted in Figure 4.4 and clearly demonstrates obstacle avoidance.

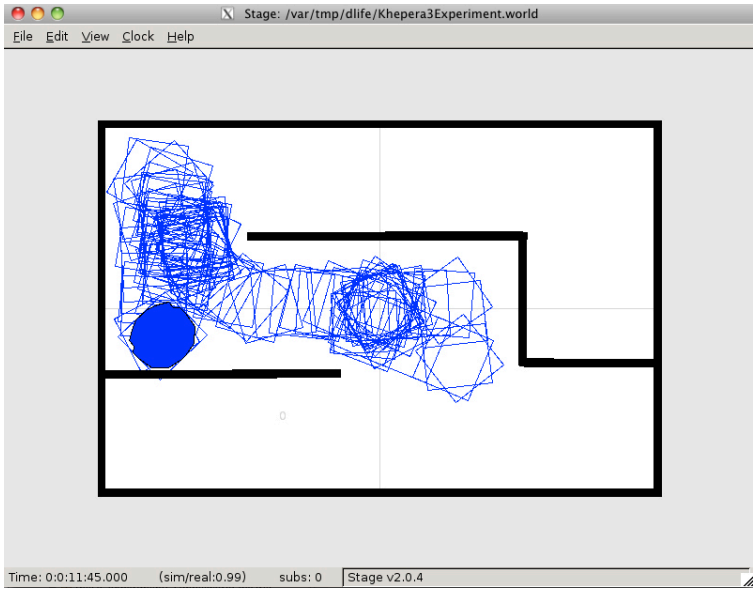

Figure 4.4: A path of the best controller running in a Player/Stage simulated world.

After just a few seconds of exploring the Q-table, the robot began to successfully navigate and explore the world. While acting sensibly, the controller was still not performing to its full potential; eventually, the robot would attempt to go closer to the wall (due to the exploration function used in Q-learning), and as a result stall for the remainder of the simulation. Since the wheel speeds are now zero, the controller is no longer able to accumulate fitness due to this one mistake. A proposed solution to this problem is discussed in Chapter 6. Furthermore, the robot moved in a somewhat jittery pattern. Only occasionally would the robot move in a smooth path. This trait is due to the coarse discretization of both the action and state spaces.

4.3.2. Transfer of the Controller to the Real Robot

With controllers performing reasonably well in simulation, it was time to make an initial attempt at moving the controller to the physical Khepera III. The simulated experiment was still too simplified to expect dramatic results (e.g. limited discrete sensors/actions and evolved in noiseless simulations). Nevertheless, several of the controllers with the highest fitness were evaluated on the physical robot. Initially, the controller was trained in simulation for 60 seconds before being transferred to the physical robot in order to build the Q-table. When the simulation training was finished, the robot was placed in a maze setup presented in Figure 4.5.

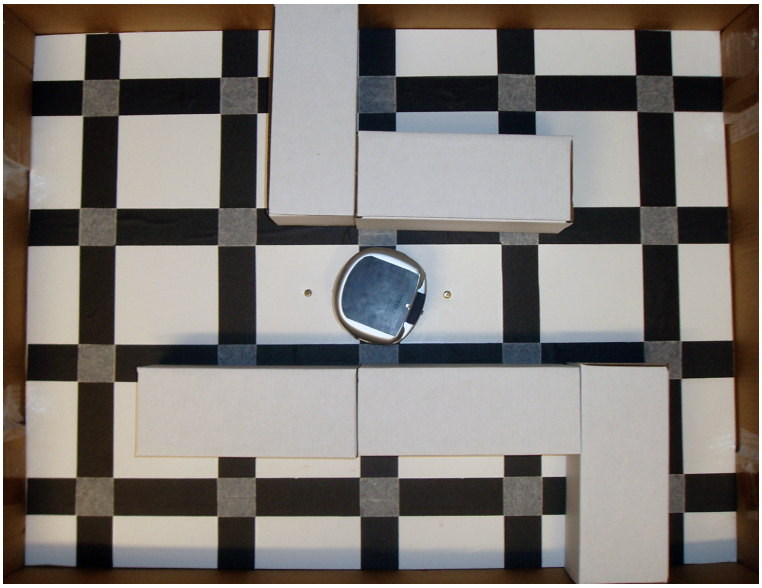

Figure 4.5: The physical Khepera 3 in a maze setup.

Considering the limitations of the simulation experiment, the robot moved somewhat effectively throughout the world. While the controller was not an optimal solution for real-world obstacle avoidance, the controller did show positive signs of intelligent behavior. It was clear that the robot would attempt to avoid walls and boxes as it drove around. Since noise was not introduced in simulation, the controller's action module (in particular the reinforcement signal) was receiving mixed information. In other words, since the reinforcement signal was evaluating a distance measurement between the desires and the resulting state, and since both state readings had a random amount of noise associated with them, this distance measurement was not reliable.

Since sensor noise is an extremely frequent occurrence in the real world, thresholds in the discrete sensor space could have been crossed as a result of the noise rather than as the result of an action. Therefore, the controller's action module began to adjust its Q-table; however, a perfect adjustment to the real world

was never accomplished. Suggestions for improving the controller's performance in the real world are discussed in Chapter 6.

4.4. The Descriptive Desires

During the research, the investigation of an additional representation of the action module's state was explored. This section presents and discusses the use of a *descriptive* state.

Ideally, a new state representation would keep as much information about the desired state of the sensors as possible while still keeping the state space small. More importantly, removing the trait of such a coarse breakup of the state was essential. Therefore, the idea was to avoid mapping sensor values to a discrete set of predetermined desired sensor values. Instead they would be mapped to a set of descriptions for each sensor. While the descriptions of the desired sensors could vary in different experiments, in the case of the robot experiment the controller could desire that each sensor be *bigger* or *smaller*. The desires network would now take in the six continuous values of each sensor (scaled and shifted to the range -2 to 2), and produce six descriptive desires (0 for smaller, and 1 for bigger) using the unipolar activation function defined in Equation 4.3. As a result, the state space was then reduced to a size of 2^6 or 64.

$$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{4.3}$$

One crucial piece of the experiment that needed to be rethought was the reinforcement signal, as the desires representation itself is very different than the

previous experiments. Therefore, the idea was to keep a count of how many sensors were able to match their desired result during each step of the controller. For example, if the desire was for a sensor to become smaller and the sensor value went down, +1 was added to the count. If the opposite happened, -1 was added to the count. At the end, the square root of this count was returned (or the negative of the square root of the absolute value if the count was negative). Furthermore, the reinforcement signal would only consider sensors that changed values. Thus, a sensor that did not become larger or smaller (e.g. the agent was following along side of a wall), would gain neither positive nor negative reinforcement.

With the new version of the state model in place, the experiment was rerun with all of the parameters from the previous simulated robot experiment kept the same.

4.4.1. The Descriptive Desires Results

Analytically, results from the descriptive desires experiment were extremely positive. A clear rise in fitness was observed early on and continued throughout the experiment for both the average and best individuals. Results from the experiment are seen in Figure 4.6.

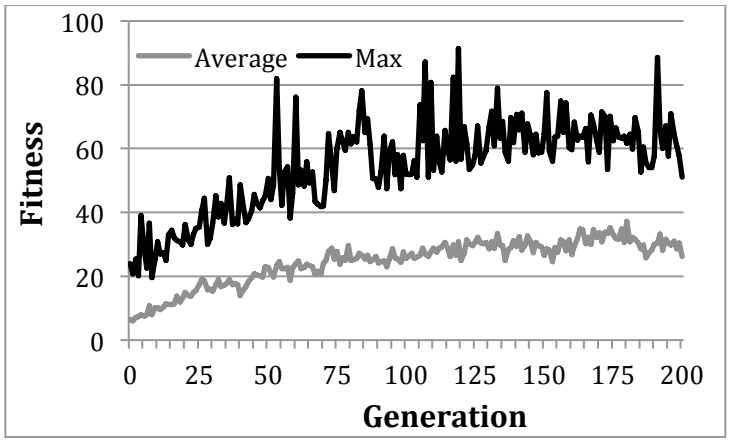

Figure 4.6: The average and best fitness values of the population over 200 generations.

It is clear that the average and best fitness values of the population where much higher than the in the previous experiment (shown in Figure 4.3). Even though the difference seemed large analytically, controllers needed to be observed in the simulator before making any conclusions. After all, controllers from the previous experiment seemed to act relatively well in simulation as well with a much lower fitness value.

As with the previous experiment, randomly selected best controllers from the final 25 generations were observed in Player/Stage. An example path of a one of these controllers is depicted in Figure 4.7.

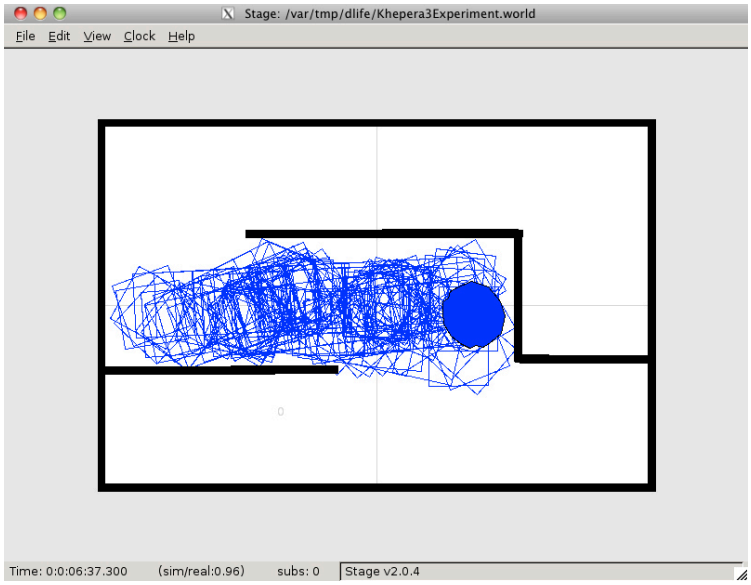

Figure 4.7: A path of the best controller running in a Player/Stage simulated world.

These best controllers from later simulations would travel around the world semi-smoothly avoiding many of the walls; however, the robot would not explore the world as much as the previous experiment's robots. Instead, the controllers would focus on the center area of the map. They would continuously wander back and forth along this section in a relatively straight path (a quality that helps accumulate high fitness). Furthermore, many of the controllers were able to continuously maintain complete obstacle avoidance for the entire 60 seconds of the trial. Thus, their fitness was being accumulated longer than the controllers in the previous experiment. While they may not be exploring much of the world, they were able to appeal to the finesses function better than the controllers from the previous experiment.

4.4.2. Transfer of the Descriptive Controller to the Real Robot

With agents performing somewhat sensibly in simulation, the controllers were tested on a physical Khepera III. As stated earlier, this test was purely a starting point for physical robot testing as the simulation experiments were still too simplified. Selections of the best controllers were trained in simulation and run on the physical robot in the maze setup presented in Figure 4.5. The robot moved very erratically throughout the world, only reacting mildly to obstacles.

The reason for this behavior is directly related to the reinforcement signal. Since the real world sensors are very noisy, they constantly have a value that is either bigger or smaller than its previous values; this is true even for a robot that is sitting still. Therefore, the counting measurement used to calculate the reinforcement signal is not a true measurement of how well the actions actually are achieving the desired states. Again, noise was intentionally left out of the simulated experiment to initially prove that the theory of the framework would work on a robot. Proposed solutions to this problem are discussed in Chapter 6.

Chapter 5

Conclusion

5.1. The Framework and *SimpleWorld* Experiments

The purpose of the *SimpleWorld* experiments was to evaluate the effectiveness of the proposed framework. By starting with the highly simplified grid-world model of the *SimpleWorld*, it was shown that the combination of this network and action module could produce a sensible agent using evolutionary techniques. To begin, it was shown that using Q-learning with the current sensor information could produce a solution to this problem within the *SimpleWorld*. This experiment served as an important step in verifying that reinforcement learning, and specifically Q-learning, was an appropriate model for use with the action module. The success of this model was clearly shown using a paired *t*-test.

The next experiments introduced both evolution and the desires network into the *SimpleWorld*. The results from both experiments showed clear improvements when the agents from earlier generations were compared to those of later generations both visually and analytically. It has been concluded that the agent was able to perform sensibly using this model.

An analysis of the differences between the three possible action module state representations was then performed. The results showed that agents that used a desires network and only used these desires when selecting an action performed the best. The desires network is working as a pre-processer of the state information to obtain the new desired state representation. Therefore, any current sensor states

that produce the same set of desired sensors will learn to perform the same action. The agent is able to focus on satisfying just these desires, rather than focusing on its current sensors. It is able to get a better sense of which actions will produce these desired states on average. Furthermore, by removing the direct relationship between the current state and the actions, it is able to better cope with non-determinism in the world. It has been shown that in a highly simplified world, the idea of the desires network and action module produce the most sensible agent out of the three methods tested.

5.2. The Simulated Robot Experiments

With the model verified to work in the *SimpleWorld*, the next step was to test its limitations within a simulated robotics domain. Noise was intentionally eliminated in this world, thus making it still somewhat simplified. As with the *SimpleWorld* experiments, this decision was made on the basis of verifying the model on increasing levels of complexity. Rather than compensating for all the complexities the simulator allowed for, this experiment would verify that the theory of the framework had the potential to work in the realm of robotics.

Discretizing the state and action spaces made a first attempt. As with the *SimpleWorld*, the desires network produced a set of desired sensor values that the controller would try to produce. This method was shown to produce a moderately successful agent in simulation. Coarse discretization of the state and action spaces was deemed to be a factor in reducing the smoothness and robustness of the robot in simulation.

One unsatisfying result of the simulation was the fact that no agent was able to achieve long-term obstacle avoidance. Before the 60-second interval had expired, robots would stall on a wall. One factor contributing to this is limitations in the update rate in the simulator. Since the agent must wait for sensor information to become available from the simulator, it does not have enough time to react to some situations. Combined with a coarse discretization of the state space, this especially becomes a problem with the exploration function. Since each agent must try each action at least ten times for each state, it would often stall before actually deciding which action may have been best. Furthermore, once an agent stalled in the simulator, it was no longer able to move around on its own. One possible solution to this problem is discussed in Chapter 6.

During the second experiment, the action module was provided with a *descriptive* desire for each sensor. This was able to accomplish a reduction in the state space as well as to be specific enough to produce sensible actions. As with the earlier experiments, results showed a clear improvement in fitness from the agents of earlier generations compared to those of later generations. This was true both visually and analytically. In this experiment, the elimination of the coarse discretization of the state space was able to produce controllers that could satisfy the fitness function better. As a result, more robots were able to run continuously in simulation without stalling on a wall for the duration of the 60 seconds.

From these experiments, it is clear that this framework can work in the domain of simulated robots. The combination of the desires network and action module has been shown to be able to produce a reasonably sensible agent.

Additionally, it has been learned that state information and reinforcement signals can be adapted and modified based on what sensor information is available or what the overall goal of the controller is. This change in state information and reinforcement signal calculation can significantly improve the performance of agents.

5.3. The Reality Gap

The main goal of this research was to investigate a new framework that could be used to bridge the reality gap. While the framework has been shown to work in the *SimpleWorld* and on a simulated robot, its performance on a physical Khepera III in the real world was relatively poor. Since the simulation was intentionally left simplified (e.g. noiseless) for initial research, this result is not unexpected. While the exact controllers evolved in the simulated experiments were not successful in the real world, it does not rule out the potential for this framework.

It is clear that such a framework can produce sensible agents. As has already been discussed, the failure in the real world is related to the discretization of the state space in the first experiment, and the lack of robustness in the reinforcement signal for the descriptive desires in the second experiment. This has been shown to be a crucial piece for the success of this framework. Therefore, appropriate modifications to this signal could overcome such a failure.

Moreover, results from the *SimpleWorld* experiment shed light on the potential for this framework to be used as a solution to the reality gap. Remember that non-determinism existed in all of the *SimpleWorld* experiments and agents that

performed the best choose their actions based purely on their desires rather than their current state. It has been shown with these experiments that removing a direct relationship between the current sensor values and the actions can help to overcome uncertainty.

This result may also relate to the way biological organisms have developed. Consider a person walking down a hallway. As they are walking, they are not directly telling themselves mathematically exactly how far away from a wall they are. Instead, their brain acts as preprocessor of this information and decides what they desire (e.g. to have their left side to be farther away from the wall). Throughout a person's lifetime, they learn what actions to take in order to satisfy these desires.

Chapter 6

Future Work and Extensions

6.1. Future Work

The investigation into the desires network and action module framework has explored a new approach in the field of Evolutionary Robotics. Much has been learned about the framework. More importantly, the potential for this framework to be used as a means to adapt to perturbations and uncertainty has been investigated. With continued efforts and research, this framework could potentially provide extremely positive results on the attempt to bridge the reality gap.

In particular, the reinforcement signal for the descriptive desires action module must be reevaluated for use on physical robots. This piece is crucial to the framework's success. Noise in the real world presents problems for the current implementation. One solution could be to weigh the change in the sensor rather than just a straight plus one or minus one count. For example, if the desire was to have a sensor become larger, and it became larger by 60 units, then this should be rewarded more than a sensor that became larger by two units. In conjunction with this approach, the desires network could also be adapted to produce quantified descriptive desires; that is, the controller could desire a *small increase* or *large decrease* as examples. As the sensor noise typically only fluctuates by a few units, this potential reimplementation of the reinforcement signal could produce a more sensible agent for the real world. A similar method could average several sensor

readings or incorporate some set threshold; in other words, only counting a positive or negative change if the sensor changed by *x* units.

Furthermore, a sensor should have the ability to stay the same or even be ignored. For example, if the robot is following a wall on it's right side, the left sensor can either stay the same or be ignored completely at the moment. Currently, the implementation requires that this sensor desires to be either bigger or smaller and no reinforcement is given if the sensor stays the same. A third value, equivalent to *stay-the-same* or *ignore* could be used to give positive reinforcement or ignore the sensor when computing the reinforcement signal in such a scenario.

Another problem seen in the experiments was dealing with the stalled robot. Once a robot clips a wall in the simulator, it would be stuck for the remainder of the simulation and thus no longer learn. As stated earlier, this problem is emphasized more by the fact that limitations in the update rate and the exploration function make this event more likely. One possible solution to this problem is to have the controller learn over several different worlds. Similar to the *SimpleWorld*, each controller can run for up to 60 seconds in a world, and then be reset and re-run in several other worlds. During these worlds, the controller could continue to learn. Once sufficient learning has occurred, the agent can be run through one final world where its fitness can be recorded. This would allow the controller to explore all possible actions without the repercussion of stalling for the entire simulation from one poor decision.

6.2 Extensions

This research has explored the evolved desires framework in the realm of Evolutionary Robotics; however, such a model could be extended further. By removing the concept of bridging the reality gap, an interesting base framework emerges for creating artificial agents that are capable of creating actions to satisfy desires based on their current state. The desires network itself could be implemented as a new module that uses its own form of reinforcement learning. Using information from its fitness evaluation during the agent's lifetime as a reinforcement signal, this module could learn over its lifetime to change its desires. It has been shown that introducing learning into an evolutionary system helps to improve the evolution itself, even if the learned information is not passed on to future generations (Nolfi, Elman, & Parisi, 1994). Furthermore, such a model could use non-evolutionary techniques if necessary and rely solely on learning methods.

The work presented opens many new avenues that are left to be explored. By continuing investigation, progress can be made to produce agents that learn to rely more on basic desires, rather than directly mapping sensors and states to actions. Such a model would help to further the development of artificial agents that mimic biological agents.

References

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

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

Coppin, B. (2004). Chapter 11: Neural Networks. In *Artificial Intelligence Illuminated* (pp. 291-326). Sudbury: Jones and Bartlett.

Hartland, C., & Bredèche, N. (2006). Evolutionary Robotics, Anticipation and the Reality Gap. *ROBIO 2006* (pp. 1640-1645). Kunming, China.

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

K-Team Mobile Robotics (2010). Khepera III. Accessed 9/14/2010: http://www.k-team.com/mobile-robotics-products/khepera-iii

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

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

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

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

Nolfi, S., Elman, J. & Parisi, D. (1994). Learning and Evolution in Neural Networks. *Adaptive Behavior*, 3(1), 5-28.

Nolfi, S., Floreano, D., Miglino, O., & Mondada, F. (1994). How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. *Artificial Life IV Conference* (pp. 190-197). Cambridge, MA: MIT Press.

Nolfi, S., & Parisi, D. (1993). Auto-teaching: Networks that Develop their own Teaching Input. In J.L. Deneubourg, H. Bersini, S. Goss, G. Nicolis, R. Dagonnier (Eds.), *Proceedings of the Second European Conference on Artificial Life*. (pp. 845-862).

Player Project (2010). Player Project Accessed 3/01/2011: http://playerstage.sourceforge.net/

Russell, S., Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

Sutton, R. S., & Barto, A. G. (1998). Chapter 1: Introduction. In *Reinforcement Learning: An Introduction* (pp. 3-23). Cambridge, MA: MIT Press.

Toris, R. (2010). Integration Between dLife and the Player/Stage Robotics Simulation System. Accessed 4/08/2011: http://users.dickinson.edu/~torisr/index.html

Walpole, R. E., Myers, R. H., Myers, S. L., & Ye, K. (2006). *Probability & Statistics for Engineers & Scientists* (8th ed.). Upper Saddle River, NJ: Prentice Hall.

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