Distributed Evolution Using Smartphone Robotics

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Travis DeVault

Major Professor: Terence Soule, Ph.D.

Committee Members: Robert Heckendorn, Ph.D.; Daniel Conte de Leon, Ph.D.

Department Administrator: Rick Sheldon, Ph.D.

May 2016

Authorization to Submit Thesis

This thesis of Travis DeVault, submitted for the degree of Master of Science with a major in Computer Science and titled "Distributed Evolution Using Smartphone Robotics," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____Date_____
Terence Soule, Ph.D.

Committee Members _____Date_____
Robert Heckendorn, Ph.D.

_____Date_____
Daniel Conte de Leon, Ph.D.

Department Administrator _____Date_____
Rick Sheldon, Ph.D.

Abstract

In learning from demonstration (LfD) a human trainer demonstrates desired behaviors to a robotic agent, creating a training set that the agent can learn from. LfD allows non-programmers to easily and naturally train robotic agents to perform specific tasks. However, to date most LfD has focused on single robot, single trainer paradigms leading to bottlenecks in both the time required to demonstrate tasks and the time required to learn behaviors. A previously untested approach to addressing these limitations is to use distributed LfD with a distributed, evolutionary algorithm. Distributed learning is a model for robust real world learning without the need for a central computer. In the distributed LfD system presented here multiple trainers train multiple robots on different, but related, tasks in parallel and each robot runs its own on-board evolutionary algorithm. The robots share the training data, reducing the total time required for demonstrations, and exchange the genetic encoding from the best solutions they've discovered. In these experiments robotic performance on a task after distributing either the genetic encoding for behavior or the demonstrations used to learn single behaviors are compared to the performance using a non-distributed LfD model receiving demonstrations used to learn multiple behaviors. Results show an improvement in performance when distributing training data on single behaviors greater than the improvement in performance when the sharing genetic information of robots trained on multiple behaviors. This implies that robots can learn robust performance of multiple part tasks by learning each of the individual parts of a task and distributing the training of the robot.

Acknowledgements

I would like to thank the lab advisers in the University of Idaho LAIR, Terence Soule and Robert Heckendorn, the many other graduate students I have worked with, Ian Tanimoto for helping with data collection when I sprained my ankle, my wife Rachel Faulkner, my mother Patricia DeVault, and my dog Zephyr.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Robotics is a growing field in industry, medicine, and the military. In the last decade robots have gone from being expensive toys to becoming household appliances of practical use, and are of great use in exploration of hazardous areas, space exploration and disaster relief. One of the largest challenges of unmanned robotic vehicles is reliable autonomous navigation.

The primary goal of this research is reduce the training time required in Learning from Demonstration (LfD) by developing multi-trainer, multi-robot learning algorithms, for the purpose of allowing robotics novices to train robots. To accomplish this research, a distributed evolutionary algorithm is used to to train a neural network to drive an unmanned robotic vehicle. The research tested this distributed method against the same learning algorithm using no distributed learning.

Autonomous robotic navigation has been an important topic of research for the past decade as the demand for mobile, unmanned robots has increased. With the use of unmanned aerial and underwater vehicles in the military, the challenges of making a robot capable of preforming important missions through mostly empty terrain have largely been met, however ground robots capable of preforming missions in a wide variety of terrains are still unsuccessful. In order for a ground robot to be able to move through an area, it must be able to determine what types of terrain it can traverse, and ideally the robot should be able to operate near people without causing injury.

As background to this research, potential solutions to two problems autonomous ground vehicles face when interacting in human environments are shown: the problem of locomotion while staying on terrain that the robot is capable of driving, and the problem of learning from humans in ways that are beneficial. To address the problem of learning from humans, a training technique used on animals known as clicker training is adapted to robotics, which allows the robots to learn to interact with their

environment through human feedback. In this research, learning from demonstration is used to train a distributed evolved neural net capable of navigating roads, outdoor environments, and bound locations.

The research in this thesis uses a distributed form of Learning from Demonstration(LfD) to provide demonstrations, used as a fitness function to evolve a neural network capable of driving an autonomous robotic vehicle. This research tests distributing the demonstrations or the fitness landscape versus distributing the genetic information of the evolved neural networks on a multi-dimensional task using an island model for distributing the genetic information. The experimental procedure created subjective data, because the robot's behavior was judged by a person observing the robot's behavior; results of the experiments showed that distributing demonstration data was more beneficial to the final robotic behavior than sharing genetic information only.

Chapter 2: State of the Art

The primary goal of this research is reduce the training time required in LfD by developing multi-trainer, multi-robot learning algorithms, for the purpose of allowing robotics novices to train robots. Research in multi-robot learning from demonstration has been very limited and focused almost exclusively on a single teacher teaching multiple robots to cooperate [12, 11, 23, 25]. This project combines ideas from several current research areas: neural networks, learning from demonstration and embodied (or on-board, on-line) evolution. Important results from these areas are summarized below.

Neural Networks

Capi, Genci, Toda, and Hideki use an evolved Neural Network to learn on-line navigation for a robotic system navigating through an indoor environment. Their goal was to make the robot learn to avoid large obstacles and they did not train the robot to identify types of ground. The robot was able to avoid walls using the trained network which was trained on-line and on-board using range finder sensors to detect when the robot had made a mistake [10]. It has been shown that in an on-line environment, improvements to evolutionary performance can occur by removing poor robotic controllers if it is possible to identify that they will perform worse than the previous controller [15]. This requires that the robot be able to identify and judge it's own fitness. The experiments performed by Haasdijk were also never performed on-board a robot.

Learning from Demonstration

In *Learning from demonstration* (LfD) a robot or other agent learns a task by observing a teacher, most often a human [4, 27]. It has been applied to a wide range of tasks with significant success, including learning gestures and movement sequences [2, 5, 9],

facial expressions [8], and communication [3]. Most important for this research, it has been used with significant success for mobile robot navigation (see for example [9, 21, 29, 28, 31]).

Sullivan, Luke, and Ziparo created a system of humanoid robots which were trained by novices using LfD to navigate themselves to a ball within a room. The robots were able to learn robust behavior that was independent of their starting points [32].

An important result from the LfD community is that the degree of correspondence between the teacher and the learner can significantly influence the speed and effectiveness of learning by demonstration and that remote control and teleoperation (in which the trainer uses the robot's "senses", e.g. the robot's camera) are some of the most effective approaches [4]. Thus, this research uses remote control of the learner as the mode of demonstration. In the experiments performed here the teacher does not use the robot's senses, but the teacher can see the robot's view if they choose to.

Chernova and Veloso created flexMLfD, a system allowing one trainer to train multiple robots [12]. In the standard approach each robot had its own training set of demonstrated behaviors and learned independently leading to heterogeneous (and cooperative) behaviors. They found that trainer workload and response time could be a significant issue, often forcing the robots to wait for additional demonstrations. As a separate experiment they tried pooling all of the training data of all of the robots (instead of each robot only using demonstrations it had personally "seen"). They found that this significantly reduced the total number of demonstrations necessary and the overall demonstration time. This strongly suggests that using a distributed approach could significantly improve training efficiency. However, it is important to note that there were a number of conditions they did not test (because this was not the primary goal of the research): all of the data was pooled, not specific subsets; training was done by one person and behaviors were not pooled - each robot learned

individually from the pool of data.

Very recently Johnson and Gonzalez developed Conceptually-based Observational Learning of Teamwork Systems (COLTS) an approach that allows multiple agents to learn to cooperate by watching a second team acting cooperatively [18]. Their teams successfully learned (in simulation) bucket brigade and pursuit-evasion behaviors. Again this shows that parallel learning from demonstration by multiple robots is possible. However, as with the previously described research, this work focused on multiple robots learning heterogeneous, cooperative behaviors rather than improving the learning process for a single behavior by distributing it across multiple trainers and robots.

There is also a history of research on *advice-exchange* between agents using, for example, Q-Learning (early examples include [13, 36, 34]). It has been shown that learning rates can be improved in cooperative problems by sharing episodes or policies [34]. Nunes et al. showed that advice-exchange between agents, including agents using evolutionary algorithms, improves learning rates [23, 22, 24]. Bongard showed that robots evolving self-models would learn accurate self-models much more quickly if they shared models [7]. These and other similar results show that in many cases exchanging advice in the form of policies for specific states can improve agent learning. However, this research did not use LfD as the source of training data and thus did not directly address the question of the effectiveness of training data from multiple human trainers.

Note that while most of the above research was conducted with LfD, the research did not use any evolutionary algorithms as the learning methods. LfD using evolutionary algorithms as a learning mechanism has been quite limited. Suwimonteerabuth and Chongstitvatana used on-line training to evolve a robot controlled by a finite state machine to remain on floors of a particular color [33]. Aler, Garcia, and Valls used evolutionary techniques and demonstrations to incrementally correct a team of

simulated robosoccer players [1]. This shows that using LfD to provide information usable by an evolutionary algorithm is a viable approach to learning for robotics.

Embodied Evolution and Distributed Embodied Evolution

In on-line, encapsulated evolutionary robotics, robots contain a population of controllers which constantly evolve during training and during run-time. Furthermore, it has been shown that better results can be obtained with a diverse population able to solve similar problems. One way to achieve diversity is to evolve several populations in different environments and for each population to share information with the other evolving populations. Distributed evolution in on-line robotics has been used as a successful learning strategy for solving navigation problems[10].

Watson et al. performed some of the earliest research using a *distributed*, *on-board*, evolutionary algorithm to evolve neural networks [35, 14]. They trained robots to perform simple phototaxis. A typical experiment used eight robots, each controlled by a Cricket microcontroller board. The robots used a simple neural network, whose weights were allowed to evolve, for guidance. The robotic agents exchanged individual genes, i.e. weights, rather than entire genomes. They found that over 140 minutes of training the agents evolved solutions that performed better than hand coded solutions.

More recently Eiben and Haasdijk et al. have performed a number of experiments on on-board, distributed evolution, although mostly using the Player/Stage (playerstage.sourceforge.net/) system as a simulation environment [15, 17, 20, 16] rather than true physical robots. They used a number of different problems including maximizing straight-line movement in a simple maze, phototaxis, and collective patrolling. They found that medium sized increases in the number of robots (e.g. from 4 to 16) significantly increased the benefits of distributed evolution, and that further improvements were observed when even larger numbers of robots were used [17].

Capi and Toda demonstrated true on-board, on-line evolution of neural controllers for robot navigation using a custom built robot [10]. Similar to the approach proposed here (see Section 5.3.3), their system partitioned collected images into regions that were used as inputs to the evolving neural controller. Using this system their robot learned to successfully navigate in indoor environments.

Summary of Previous Research

The primary goal of this research is multi-trainer, multi-robot learning. To achieve this goal the above research shows the following:

Neural networks are effective at controlling robots

Neural networks have successfully been evolved online in embodied robots

Learning from Demonstration is a common and effective learning technique for allowing novices to teach robots

Learning from Demonstration has been effective in allowing one teacher to train multiple robots

Learning from Demonstration has been used in evolutionary algorithms to provide information used for learning

Distributed evolutionary algorithms have been shown to increase the learning rate of multiple robots

This research accomplishes it's primary goal by building on the above successes to create an evolved neural network capable of controlling an autonomous robot. The research shows that one or more trainers can train a robot, and by using LfD to provide a viable fitness function, the robot can evolve to complete complex tasks. Furthermore, this research shows that robots can distribute both genetic and demonstration information, effectively allowing one trainer to train multiple robots, and allowing a robot to learn from multiple trainers.

Chapter 3: Hardware and Robot Information

This research was designed to used embodied on-line evolution using the University of Idaho COTSBOTs technology developed by Terence Soule and Robert Heckendorn. The COTSBOT is a Commodity Off The Shelf roBOT, designed to be a computationally powerful and economically viable research robot for university laboratories.

The robots consist of three basic components: an Android smart phone, an Arduino micro controller, and a robot platform. A typical smart phone used in these experiments is the HTC Desire, which has a 1GHz Qualcomm Snapdragon processor, with 768 MB of RAM, and a Qualcomm Adreno 205 GPU. The smart phone sends commands via bluetooth to the Arduino microcontroller, which in turn drives the motors on the robotic platform, a Rover 5 Tank chassis[1], which is a small (22.5cm by 24.5cm), tracked vehicle controlled by two DC motors and capable of running continuously for several hours while carrying the micro controller and smart phone.

The decision to use an embodied COTSBOT was made at the beginning of the research in acknowledgement of the difficulty of making a robot behave the same in the real world as a robot would behave in a simulation. Using a COTSBOT allowed the research to make use of a wireless network to communicate to a desktop computer. The desktop computer communicated with the robot during the research to handle the distribution of information between robots. No external Arduino sensors were used during these experiments, although they could be used for future work.

[1]Rover 5 Tank chassis http://www.dfrobot.com

Chapter 4: Preliminary Work

This chapter shows two pieces of preliminary research done as a precursor to the main work presented in this thesis. The first research uses a technique called Clicker Training to allow a novice to train a robot to preform specified tasks. The second work is research using a distributed system of robots learning to navigate safely on a particularly colored road.

4.1 Clicker Training: Reinforcement learning for a mobile CotsBot

This research was done at the University of Idaho with the goal of creating and testing robotic controllers capable of allowing novice robot trainers to train robots to be able to complete tasks with multiple parts. This research was done by several graduate students: Nathaniel Eble, Juan Marulanda, Jayandra Pokharel, Joshua Rubini, and myself. Since the work was conducted with so many people, and because the research does not directly lead to the primary research in this thesis, an abbreviated introduction from the paper written for this research is included here. It is important because of it's relationship to the research goals of the primary research in this thesis: creating robots which are easily trained by people whom are not experts in robotics.

4.1.1 Introduction

Intelligent, autonomous, robotic agents are desirable because of their potential application to a wide variety of fields, such as natural disaster response and exploration of extraterrestrial landscapes [6, 37]. For complex, and dynamic applications such as these, it is difficult to design effective controllers before deployment, and it is therefore desirable for a controller to learn while it is deployed.

Robotic learning can take several forms including: supervised, unsupervised, and reinforcement.

Supervised learning strategies involve telling an agent what it should do given a set of inputs and then using that information to generalize behaviors for an inputs. An example of this type of learning is using back propagation to train a neural network.

Unsupervised learning attempts to cluster unlabeled data without any feedback. Common approaches include hidden Markov models and principle component analysis.

Reinforcement learning strategies include guiding an agent towards desired behaviors and then reinforcing those behaviors in some way to influence their chances of happening in the future.

We present two implementations of clicker training, which is a form of reinforcement learning, as a means of training a mobile, autonomous robot. The first method uses a look-up table, and the second use a learned grammar. Clicker training is the approach commonly used to train pets, but it can be easily expanded to the training of any agent; including robots. In this method, a robot learns a behavior through trial-and-error interactions with a trainer [19]. The results of these interactions are reinforced in an effort to increase or decrease their probability of occurring in the future based on whether a positive or negative reinforcement was received. Training in this fashion leads to robots that fall within the *human acting* category of the Artificial Intelligence description in [26], meaning that the robot behaves in a way that a human would.

The controllers were implemented on a mobile robot built using Commodity-Off-The-Shelf design principles (COTSBots). Reinforcements were given to the robot by clicking either a positive or negative reinforcement button on a connected smartphone. The robot starts by taking random actions and then pausing for one second to allow the trainer to provide reinforcement. If reinforcement is given, the controller increases

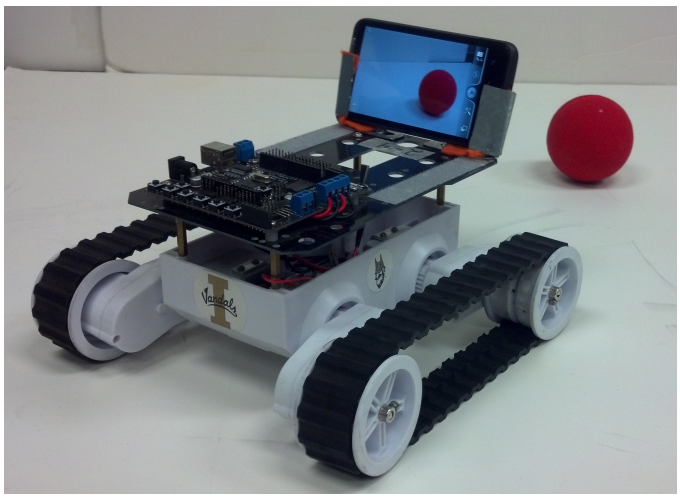

Figure 4.1: Robot designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles. The robot consists of three basic components: a smartphone, an Arduino microcontrollers, and a mobile robot platform.

or decreases the probability of the previous action occurring if the robot encounters the same state, or set of inputs, based on whether positive or negative reinforcement was received. After training, the robots could operate autonomously, where no further reinforcements are given. Testing showed that clicker training, using either control method, was an effective way of training a robot to complete a task, and training multiple times has a significant impact on the performance of the grammar method.

Clicker Training expands on previous applications of reinforcement learning to robotics in an effort to show that a simple training system can be developed that allows a robotics layperson to train robots for a particular task within the trainer's own field of expertise.

4.2 On-Board, On-line Distributed Evolutionary Learning by Demonstration

This research was done at the University of Idaho in 2014, and submitted to the Genetic and Evolutionary Computation Conference (GECCO) for 2014. The paper was rejected, but lead to the creation of the primary research in this thesis. As with

the previous section, this work is preliminary to the research of this thesis. An edited version of the paper is presented here, with some of the redundant information (robot hardware etc.) removed.

4.2.1 Introduction

In LfD the learning process is typically on-line, i.e. the agent is learning during the demonstrations so that it can perform the task immediately, or at least shortly, after the demonstrations are complete. However, as the complexity of the behavior increases so does the learning time required, which makes it more difficult for agents to learn behaviors as they are demonstrated. This can be a particular issue in LfD because the process is often cyclic: the trainer demonstrates a task or behavior, the agent attempts to replicate it, and, depending on the level of success, the trainer can chose to reinforce the task with additional demonstrations or move on to related tasks. This has significantly limited the use of evolutionary computation (EC) in LfD applications in the past because EC is widely viewed as too slow of a learning mechanism (despite other advantages) to fit into the LfD cycle.

Additionally as LfD is applied to more complex behaviors, larger and more complex training sets of demonstrated behaviors are required and search spaces become more complex. Both of these features make it much more likely that an evolutionary algorithm will become stuck in a local optima.

To overcome these two difficulties: longer training times and more difficult search spaces; we present a *distributed*, evolutionary LfD approach in which multiple trainers train multiple robots in parallel, each robot runs its own on-board evolutionary algorithm and exchanges elite individuals (wirelessly) with the other robots, thereby creating a distributed island model. Showing that such an approach is feasible significantly increases EC's suitability as a learning technique for LfD by mitigating issues

of efficiency of search.

Our results show that distributed evolution produces behavior that is more robust and outperforms behavior which is evolved with only one trainer and training track (path). The robots which evolved without distribution performed about as well as a two robots evolving with different trainers with only half the generations of evolution time.

4.2.2 Methods

The trainer has two methods of providing instructions to the robot, the first is a hand-held remote control which is another smart phone communicating with the robot via bluetooth. The second is a server running on a PC, which the robot connects to via Wi-Fi. Both the server and the remote can give the robot direction as to what state the robot should be in, and what action the robot should take.

There are four states that the robot uses during this experiment:

Remote Operation: In this state the robot can receive drive instructions from the remote or server. The robot will move, but it will not train it's vision, nor will it collect training cases for evolving behaviors. This is the robot's starting state and is only used when no other states are desired.

Vision Training: In this state the robot will receive drive instructions from the remote or server. When the robot moves, it will train it's vision, adding color histograms examples identified as being the road. As seen in Figure 4.2, there is a small region in the robot's view, denoted by the red box, which the robot assumes to be the safe path while in this state. Color histograms are taken from this region and added to the library of safe road examples. At the start of training the trainer drives the robot on the path for ten drive commands giving the road library ten examples of known road. This amount of vision training, consisting of ten samples, has been chosen by trial-and-error as a sufficient amount of training to recognize safe roads.

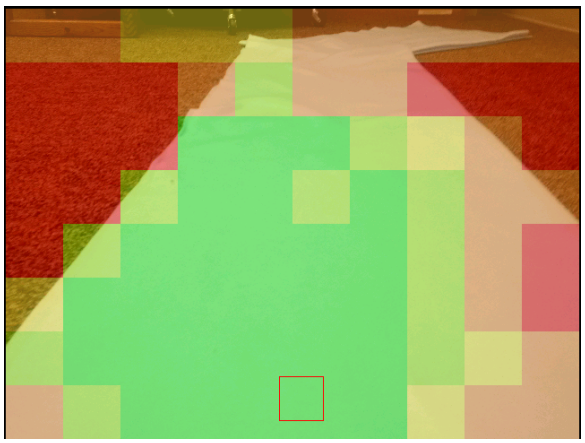

Figure 4.2: Robot's screen colored to indicate calculated road values. The box at the bottom of the screen is the region from which the road library is built. It is assumed that the box never leaves the desired road type so each region can be assumed to be an example of a safe road type. The screen is divided into 40 regions and each region is assigned a probability of being a known road type. These 40 values then become inputs to the controlling neural network.

Learning from Demonstration: In this state the robot receives drive instructions from the remote or server. When the robot receives an instruction, it creates a *Training Case* for the artificial neural network (detailed description in 4.2.4). A training case consists of the image the robot sees, and the action taken. At this state the evolution also starts, which is described in 4.2.4.

Autonomous Operation: In this state the robot does not receive commands from the trainer. The robot evaluates it's vision and chooses to take an action on it's own. It does this using the ANN trained during the Learning from Demonstration state. Performance during autonomous operation is used to determine how successful training was.

4.2.3 Experiment Description

In order to show evolutionary computation's suitability in a LfD system, we create an experiment with three different trials: in the first trial is on-line, but not distributed, the robot is trained for 100 generations; the second trial is a distributed trial with

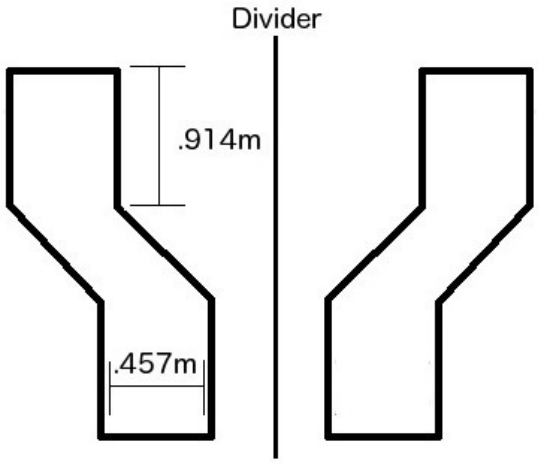

Figure 4.3: Training Tracks

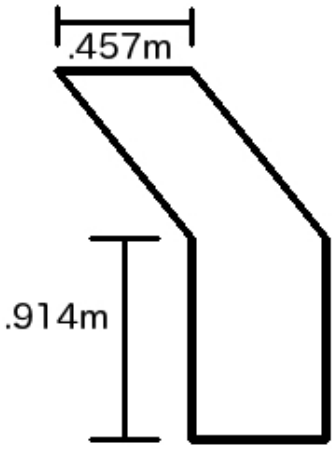

Figure 4.4: The 45° testing track. This track is designed to test the robot's ability to navigate the same type of corner that it was trained on.

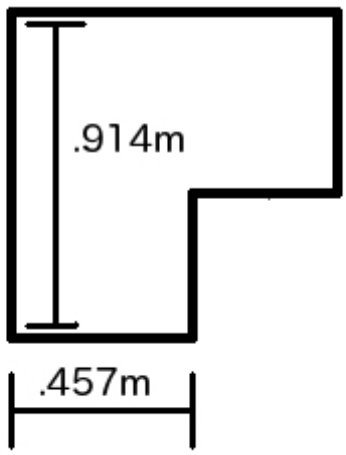

Figure 4.5: The 90° testing track. This track is designed to test the robot's ability to generalize its learned knowledge of how to complete a 45° corner to the completion of something it was not trained to navigate.

two robots, each of the two robots is trained for 50 generations; the third trial is a distributed trial with two robots, each of the two robots is trained for 100 generations. This is done so that the experiment's results compare two distributed robots training for 50 generations to a single robot training for 100 generations. Additionally, the results can compare two distributed robots training for 100 generations to the single robot training for 100 generations.

At the start of each trial, the robot is driven by a trainer on a road of distinctly colored felt. The trainer puts the robot into a Vision Training state and drives the robot for ten steps. This allows the robot to build a library of road color histograms, which the robot then uses to distinguish road and non-road.

During learning from demonstration phase a human drives the robot, following a path defined by colored felt on a differently colored background. The human uses a remote control with discrete forward, left, and right options. While the human is driving, the robot is building a training set of road images paired with the correct, i.e. driver supplied, action.

In all trials, the evolutionary processbegins after one of each command (Left, Right, and Forward) have been given. To begin the trial, the robot was placed on

the track shown in Figure 4.3 in front of the corners and given a single left and a single right training case. The robot was then placed at the start of the track and driven to the end of the track, and picked up and turned around, to drive back to the start. This was repeated for as long as evolution ran (50 or 100 generations). In this process, evolution starts when the robot receives it's first forward command.

Note, Figure 4.3 shows that for the distributed evolution the two robots are not being trained on the same track, because the track is mirrored. This means that for one robot the first turn is a right turn instead of a left turn, and that visual inputs, and especially background images, will be different between the two robots.

In the two distributed cases, the robots are trained in parallel, each with it's own trainer. The trainers give the first forward command at the same time, starting the evolution. After the first command, the trainers do not stay synced, they simply train the robot independently. The robots share their best individual in each generation with the server. The server waits for a complete list of the best individuals, and sends the list to the connected robots. The robots then add these individuals to their population. No mechanism is used to keep the evolution in lockstep, so if one robot is evolving slower, it is possible for the other robot to evolve a few generations without receiving the best individuals from the server. In these experiments the evolutionary process did get out of sync, but never by more than 5 generations.

In the training stage an evolutionary algorithm is run on the robot (in the smart phone) to train a neural network to generate the correct action for each image. The evolutionary algorithm runs for 50 or 100 generations and the individual with the best fitness is then tested in autonomous mode. In the parallel, distributed case only one robot is tested, using its best evolved behavior.

During autonomous operation the robot uses its best evolved neural network to follow the test tracks autonomously. In the robot, input images are continuously compared against the road library to generate road values as inputs for the neural

network which then returns an action. Performance is measured by the total number of turns successfully navigated on the test tracks.

The trained robot drives autonomously on each of the two testing tracks three times. The robot the reverses the starting direction on the testing tracks, and once again attempts to drive autonomously the track three times. This gives a total of twelve data points for each trial. Failure of the robot is defined as the robot driving off the track or by the robot moving back toward its starting location.

4.2.4 The Genetic Algorithm

This section describes the Neural Network architecture, the Evolutionary Algorithm, and the way distribution is handled for the experimental cases that use distribution.

4.2.5 Neural network architecture

The neural network has forty input nodes, corresponding to the forty image cells4.2, a single hidden layer consisting of five nodes, an output layer with three nodes, and eight bias nodes. The nodes all use a sigmoid activation function. The input nodes for the network come from the image received from the smart phone camera. The trained vision algorithm breaks the image into an 8×5 grid of cells, and compares each cell in the image to the library of histograms containing safe road colors. The vision algorithm assigns a probability to each cell representing the likelihood that the cell contains road. The histogram comparison is done using the OpenCV image library for Android. These forty values are the inputs to the neural network.

The neural network's output layer has three nodes, representing the action that the robot should take. The three values correspond to the forward, left, and right actions. The genetic algorithm uses fitness pressure to make losing values move far away from the winning value as discussed below. The robot will always move in the

direction of the winning value. The three output values are real numbers between -1 and 1, so while ties are possible, they are very rare and never occurred during this experiment.

4.2.6 The evolutionary algorithm

We used a standard generational genetic algorithm (GA) to evolve the neural network weights. Individuals are an array of values defining a complete multilayer perceptron neural network; each individual has 223 weights (40 input nodes $\times$ 5 hidden nodes = 200, and 5 hidden nodes $\times$ 3 output nodes = 15 and 8 bias nodes). Initially the weights were randomly set in the range -1.0 to 1.0. An elitism of one is used.

Selection was performed via a tournament of size three. Two individuals are selected as parents and produce only one offspring. Uniform crossover was used. Each offspring has 223 weights which are mutated at a rate of 0.2, on average resulting in 44 mutations per offspring. The amount of mutation was uniformly distributed between -0.1 to 0.1.

Fitness was determined by comparing the cases collected in the training data to the actions taken by each individual network with the same training case. The training cases are divided into categories based on the expected action (Left, Right, Forward). The cases are weighted so that each category of action (Left, Right, Forward) is worth the same amount of fitness. The fitness is calculated this way because during training, there are generally ten forward commands to every one turn command given. The algorithm for the fitness function is presented in Algorithm 1. If the network chooses the correct action, it always gets a positive fitness for that training case, but the fitness will increment more if the difference is at least equal to the threshold. The threshold used in this algorithm is 0.4, which was derived from trial-and-error.

The other parameters of the evolutionary algorithm are summarized in Table 4.1.

Algorithm 1 Genetic Algorithm Fitness Function

```
for each network in population do
    fitness = 0
    for each training case do
        actualResults = network result from running training case
        expectedResults = results from training case
        winningValue = actualResults at the position of the expectedResults winner
        difference1 = winningValue - actualResults at first position other than winner
        difference2 = winningValue - actualResults at second position other than winner
        for each difference do
            if difference ≥ threshold then
                fitness += 0.5 / number of training cases in that catagory
            else
                fitness += 0.5 * (difference / threshold) / number of training cases in that catagory
            end if
        end for
    end for
    fitness = 1 - fitness / 3
end for
```

Population size	40
Generations	50 or 100
Mutation rate	0.2
Crossover rate	100% Uniform
Selection method	Tournament(size 3)

Table 4.1: Summary of the evolutionary algorithm parameters.

4.2.7 Distribution

While the genetic algorithm is running, after each generation the algorithm sends it's best individual to the server. The server continues to listen until it has received an individual from every robot connected to the server. When it has received an individual from every robot, it sends the whole set of individuals to every robot.

When the robots receive the set of individuals, the robots put the individuals into a special portion of it's population. This portion of the population can be used for selection, but it is not overwritten with the next generation. This sub-population is used so that the distributed individuals do not get overwritten quickly. This portion of the population is only overwritten by the next set of individuals sent to the robot from the server. This portion of the population is also not used when determining the robot's average fitness. After the robot receives those elite individuals from the server, it calculates the fitness of the elites using it's own set of training cases, and the individuals can be selected using the normal selection process. Note that one of the individuals received is the elite individual from the robot's own population, giving the robot an effective elitism of two.

4.2.8 Results

Tables 4.2 and 4.3 summarize the results of these experiments. The first column describes which type of experiment the data was gathered from. The second column lists the training instance, recall that for each experiment the robot was trained five times and tested three times. The remaining columns show how many times the robot successfully navigated a particular corner, and the total number of successfully navigated corners.

First, the results show that parallel training does significantly reduce the total

Table 4.2: The number of successes each trial and training over the course of the four test corners. The total successes of the 50 generation distributed trial an the 100 generation single trial were very close. The 100 generation distributed trial outperformed both.

	Training	45° Left	45° Right	90° Left	90° Right	Success out of 12
50 Dist	1	0	2	0	3	5
	2	1	2	2	2	7
	3	1	2	0	2	5
	4	2	3	2	2	9
	5	0	1	2	2	5
100 Dist	1	0	2	1	1	4
	2	2	3	2	2	9
	3	1	3	3	3	10
	4	1	2	1	3	7
	5	1	2	3	3	9
100 Gen	1	1	1	2	0	4
	2	1	2	1	2	6
	3	3	3	2	2	10
	4	1	1	2	2	6
	5	0	1	0	3	4

Table 4.3: The average and best fitness of each round of training. The number of Train Cases and the number of times individuals were shared in the distributed cases.

	Training	Avg	Best	Training Cases	Shared Ind.
50 Dist	1	0.53	0.4	16	47
	2	0.44	0.27	16	47
	3	0.2	0.16	16	49
	4	0.31	0.15	16	49
	5	0.61	0.46	17	46
100 Dist	1	0.46	0.21	31	94
	2	0.2	0.07	39	84
	3	0.2	0.08	33	97
	4	0.52	0.39	31	96
	5	0.33	0.03	33	96
100 Single	1	0.23	0.2	26	0
	2	0.13	0.11	31	0
	3	0.05	0.03	33	0
	4	0.03	0.02	33	0
	5	0.25	0.23	36	0

training time required, because the performance is measured by the number of successfully navigated corners, and the 50 generation distributed experiment results were nearly the same as the results of the 100 generation without distribution. The 50 generation distributed robot navigated 51% of the corners, where the 100 generation robot without distribution completed 50% of the corners. The 100 generation distributed robot navigated 65% of the corners correctly.

These results strongly suggest that having multiple trainers, with potentially different driving styles, does not make the learning process more difficult - e.g. by creating conflicting training cases, which is shown by the nearly identical results from the 50 generations distributed experiment and the 100 generation with no distribution experiment.

Based on previous research with island model evolutionary algorithms, it is likely that the distributed approach will also make it less likely that the evolutionary process will become stuck in local optima. However, this can not be definitively concluded from the current experiments.

Table 4.2 shows a clear improvement in performance when distribution is used, as the robots evolved for 50 generations were able, on average, to outperform the robot evolved without distribution for 100 generations. This suggests that the differences in the training tracks, and the differences in the driving styles of the trainers may have had an impact on the performance of the robot.

Although Table 4.3 shows no clear increase in the rate of fitness growth for the average individual of the population, the best individuals have better fitnesses in the distributed cases, with three of the five trials having a fitness of less than 0.1.

4.2.9 Conclusions

Our results show that distributed LfD, using multiple trainers and multiple robots, is a practical approach to reducing the time required to produce sufficiently

rich training sets for learning. In addition, leveraging multiple robots to create a distributed, island model, evolutionary algorithm significantly increases the overall rate of evolution.

Using the distributed island model performance, was significantly better than performance of the non-distributed model. This suggests that the evolved behaviors from different training cases and drivers were helpful in creating a more robust evolved behavior.

One surprising result is that the 50 generation distributed robot performed just as well as the 100 generation non distributed robot. The robot running for 100 generations had more training cases, and more time to evolve. It's fitness was significantly better than the distributed, as would be expected, but the performance was not significantly better. This suggests that performance on the track is not as closely tied to fitness (as measured by correct decisions), as might be assumed. The ability to handle more situations is superior to the ability to handle a few training cases very well, so performance is the better measure of success, which will also lead to more generalized solutions.

4.2.10 Analysis

The proceeding two pieces of research lead to the primary research presented in the next chapter. The work with distributed robot road following lead to the decision to use performance as the measure of success for the experiments involving distributed evolutionary LfD. However, it was discovered that the vision system using a library of histograms was not very robust, so a new vision method is used as well as a slightly different neural network and fitness function.

Chapter 5: Learning from Demonstration for Distributed, Encapsulated Evolution of Autonomous Outdoor Robots

This chapter represents the main body of work for this thesis. This chapter was written as a paper submitted and accepted as a poster to GECCO in 2015. The research presented in this chapter is unedited from that submission.

5.1 Abstract

In learning from demonstration (LfD) a human trainer demonstrates desired behaviors to a robotic agent, creating a training set that the agent can learn from. LfD allows non-programmers to easily and naturally train robotic agents to perform specific tasks. However, to date most LfD has focused on single robot, single trainer paradigms leading to bottlenecks in both the time required to demonstrate tasks and the time required to learn behaviors. A previously untested, approach to addressing these limitations is to use distributed LfD with a distributed, evolutionary algorithm. Distributed on-board learning is a model for robust real world learning without the need for a central computer. In the distributed LfD system presented here multiple trainers train multiple robots on different, but related, tasks in parallel and each robot runs its own on-board evolutionary algorithm. The robots share the training data, reducing the total time required for demonstrations, and exchange promising individuals as in typical island models. These experiments compare robotic performance on a task after distributing the behaviors or the simple demonstrations to performance using a non-distributed LfD model receiving complex demonstrations. Our results show a strong improvement to behavior when using distributed simple demonstrations.

5.2 Introduction

In this paper we present a distributed, evolutionary approach to learning from demonstration (LfD) for teaching mobile robots. In LfD [1] a trainer performs the target actions and the agent records the both the current state and the demonstrated actions to build a training set of state-action pairs that it learns from. In most, but not all, LfD research the trainer is a human. Thus, one advantage of LfD is that it allows a human trainer with no programming skill to quickly retrain a robot to perform a novel task or operate in a novel environment.

In LfD the learning process is typically on-line, i.e. the agent is learning during the demonstrations so that it can perform the task immediately, or at least shortly, after the demonstrations are complete. However, as the complexity of the behavior increases so does the learning time required, which makes it more difficult for agents to learn behaviors as they are demonstrated. This can be a particular issue in LfD because the process is often cyclic: the trainer demonstrates a task or behavior, the agent attempts to replicate it, and, depending on the level of success, the trainer can chose to reinforce the task with additional demonstrations or move on to related tasks. This has significantly limited the use of evolutionary computation (EC) in LfD applications in the past because EC is widely viewed as too slow of a learning mechanism (despite other advantages) to fit into the LfD cycle.

Additionally as LfD is applied to more complex behaviors, larger and more complex training sets of demonstrated behaviors are required and search spaces become more complex. Both of these features make it much more likely that an evolutionary algorithm will become stuck in a local optima.

To overcome these two difficulties: longer training times and more difficult search spaces; we present a *distributed*, evolutionary LfD approach in which multiple train-

[1] Also known as learning by demonstration and closely related to imitation learning and apprenticeship learning.

ers train multiple robots in parallel, each robot runs its own on-board evolutionary algorithm and exchanges elite individuals (wirelessly) with the other robots, thereby creating a distributed island model. Showing that such an approach is feasible significantly increases EC's suitability as a learning technique for LfD by mitigating issues of efficiency of search.

Our research focuses on how to train robots in a noisy, real-world environment. Two types on distributed evolution are tested. In the first, robots are trained in specific conditions and their behaviors are distributed in the form of migrated individuals. The second case distributes the training demonstrations by taking the same training data trained in specific conditions, and combines it to create a large demonstration set.

The evolved behaviors are tested in an environment similar, but different than their training environment using physical robots. The robots were observed by an outside individual that did not train the robots, and given a grade between 1-5 as to how well the robots demonstrated specific behaviors.

Our result show that, even in a simple environment, distributed training cases targeting specific conditions to the evolved agent produces more robust behaviors than an evolved agent that is given more noisy demonstrations in a full environment. Distributing behaviors is not effective in creating learned behaviors outside of the behavior set that is used in the fitness calculation.

5.3 Methods

The robots used in these experiments were designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles [30]. The robots consist of three basic components: an Android smart phone, an Arduino micro controller, and a robot platform. A typical smart phone used in these experiments is the HTC Desire, which has a 1GHz Qualcomm Snapdragon processor, with 768 MB

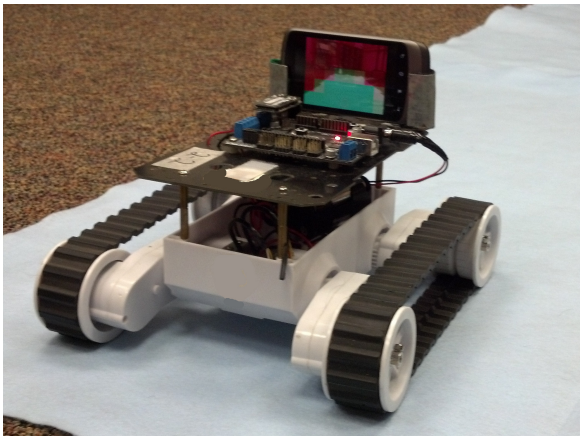

Figure 5.1: The robot. The smart phone receives wireless commands from the user and forwards the commands via bluetooth to an Arduiono type microcontroller that controls the robot's motors. The smart phone can perform image processing and on-board learning.

of RAM, and a Qualcomm Adreno 205 GPU. The smart phone sends commands via bluetooth to the Arduino microcontroller, which in turn drives the motors on the robotic platform, a Rover 5 Tank chassis[2], which is a small (22.5cm by 24.5cm), tracked vehicle controlled by two DC motors and capable of running continuously for several hours while carrying the micro controller and smart phone. The complete robot is shown in Figure 5.1.

There are two states which the robot uses during this experiment:

Learning from Demonstration: In this state the robot will receive drive instructions from the remote or server. When the robot receives an instruction , it creates a *Training Case* for an Artificial Neural Network). The training case being: the image the robot sees, and the action taken. At this state the evolution also starts when doing on-line evolution. For offline evolution, the training cases are saved, and evolved later.

Autonomous Operation: In this state the robot does not receive commands from the trainer. The robot evaluates it's vision and chooses to take an action on it's own. It does this using the ANN trained during the Learning from demonstration

[2]Rover 5 Tank chassis http://www.dfrobot.com

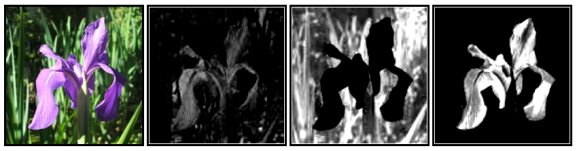

Figure 5.2: Iris image with the Red, Green, and Blue components enhanced

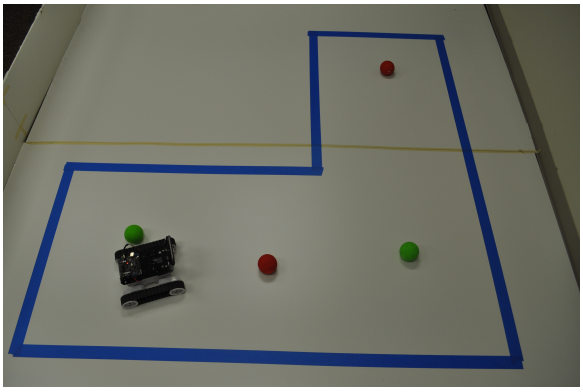

Figure 5.3: The robot was trained by collecting demonstrations in this training environment. For demonstrations requiring only red and green balls, the robot was positioned to not see the blue tape.

state.

This experiment uses an Artificial Neural Network as the robotic controller. In the beginning, the neural network does not control the robot, the human trainer does. While training, the robot processes visual data and the actions the trainer makes as demonstrations of how to behave in a given situation. The robot can be set to train in online, meaning that the evolution will begin as soon as the trainer gives demonstrations. the robot can also be trained offline, where demonstrations are collected, stored, and evolved separately. The robot can then load an evolved network and act autonomously. For these experiments, offline training was used.

5.3.1 Demonstrations

Demonstrations are composed of the input the neural network would receive and the action taken by the trainer. Demonstrations are collect when the trainer puts the robot into LfD mode and then drives the robot by giving it discrete commands from

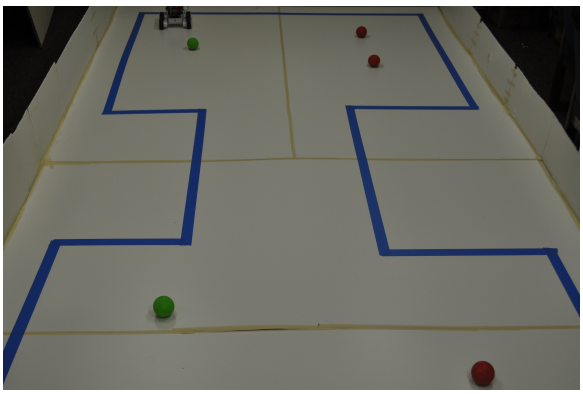

Figure 5.4: The robot was tested in an environment different from it's training area. During the testing phase, the robot autonomously navigates the testing environment while human observers watch the robot's behavior.

the remote control. The commands that the trainer has access to are: Left, Forward, Right, Backward, and Stop.

The input the neural network receives is visual input from the phone's camera, section 5.3.3 provides a detailed description of the neural network and the genetic algorithm used to evolve the network. The camera's image is divided 8×5 into 40 regions. The regions each send three real values as neural network inputs. The regions average their Red, Green, and Blue values, and then modify the value to be acceptable neural network input. The enhancement formula is (X-.5(Y+Z))/255. Figure 5.2 shows how an enhanced image's primary color components stand out. With each region sending 3 values for each region, the neural network receives 120 values as input.

Figure 5.3 shows an image of the training environment used to collect demonstrations. The trainer was allowed to drive the robot around the environment, or pick up the robot and place it to give the robot a specific demonstration.

5.3.2 Experiment Description

In order to show distributed evolutionary computation's suitability in training a LfD system, we created an environment to test the robots behaviors in motion, and in

interaction with objects. The environment have borders which the robot is trained not to cross, and different colored balls which the robot was trained to search for and stop at once the robot moved close enough to the ball. Behaviors are defined here.

Green Ball Interaction The robot tries to center the green ball on the phone's screen. If the robot can see the green ball far away, it will move towards the ball while keeping the ball centered.

Green Ball Stop The robot will stop when the green ball is on the screen and the robot is close to the ball.

Red Ball Interaction The robot tries to center the red ball on the phone's screen. If the robot can see the red ball far away, it will move towards the ball while keeping the ball centered.

Red Ball Stop The robot will stop when the red ball is on the screen and the robot is close to the ball.

Blue Tape Avoidance The robot will not run over the blue tape.

Our experiment uses three types of evolution.

Evolution 1: These controllers are evolved with no distribution. The robot is trained given demonstrations including the entire training environment. The robot is able to see borders, and different colored balls together.

Evolution 2: Three robots are trained on individual behaviors. One robot was only given demonstrations with the red balls, another robot received demonstrations involving the green balls, and the last robot was only given demonstrations with the blue border of the environment. Robots in these trials were evolving by distributing their best individuals during every generation of evolution. Evolution was also done on these demonstrations without using distribution to allow analysis of the behavior.

Evolution 3: Demonstrations for these robots were collected by combining the demonstration data from *Evolution 2*. This is the case of distributed demonstration collection. The robot does not receive behavior data from other robots.

In Evolution 1, the robot was given 90 demonstrations using the training environment as seen in figure 5.3. In Evolution 2, the robots were only given 30 demonstrations, because the behaviors are distributed, and we did not want to give the robots in this case the advantage of additional information. Evolution 3 each robot forms it's demonstration set from the 30 demonstrations from each of the three robots in Evolution 2, thus Evolution 3 robots also have 90 demonstrations to learn from.

In the demonstration phase, the trainer collects demonstrations of the robot navigating the testing environment. The trainer was instructed not to allow the robot to drive over the blue tape, to drive the robot toward red and green balls, and to stop the robot when the robot got close to the red and green balls.

To test the robot's behavior we used a blind survey similar to how behavioral scientist judge the behavior of animals in the wild. A human observer was gives a description of the behaviors the robot was trained to perform, and then without know what type of training was used, score the robot's behavior between 1-5. 1 meaning that the robot did not appear to demonstrate that behavior at all, and 5 meaning the robot demonstrated the behavior every time, in multiple positions.

The robot was placed in the testing environment (Figure 5.4) and the observer was allowed to watch the robot's behavior for 2 minutes. During this time the turned off and put back into the blue border if it wandered outside of it. The observer was allowed to physically move the robot if they felt the need to observe a specific behavior, such as placing the robot near a ball to see if the robot learned to stop. Five trials were done with each of the evolution types.

Table 5.1: Each type of evolution was trained five times and evaluated in a blind survey. The robot was scored 1-5 on how it behaved in the testing environment by how it interacted with the environment. This table shows the evolutionary cases where distributed evolution was used.

Evolution Type	Trial	Green ball find	Green ball stop	Red ball find	Red ball stop	Avoid blue tape
Distributed Training Cases	1	3	4	3	5	5
	2	3	2	4	5	5
	3	3	5	5	5	5
	4	2	2	5	5	2
	5	4	4	2	4	5
Distributed robot trained on green balls	1	5	5	1	1	4
	2	4	5	3	1	3
	3	5	5	1	1	2
	4	5	5	1	1	4
	5	4	4	1	1	2
Distributed robot trained on red balls	1	1	1	5	5	2
	2	1	1	5	5	3
	3	1	1	5	5	2
	4	1	1	3	4	3
	5	1	1	5	5	2
Distributed, trained on blue tape	1	1	1	1	1	4
	2	3	1	2	1	5
	3	1	1	1	5	5
	4	1	1	1	1	5
	5	1	1	1	1	4

Table 5.2: Each type of evolution was trained five times and evaluated in a blind survey. The robot was scored 1-5 on how it behaved in the testing environment by how it interacted with the environment.This table shows the evolutionary cases where no distribution in the evolution was used.

Evolution Type	Trial	Green ball find	Green ball stop	Red ball find	Red ball stop	Avoid blue tape
No distribution, all behaviors	1	4	5	1	2	3
	2	5	5	4	5	5
	3	1	1	2	3	5
	4	1	4	1	3	5
	5	4	3	4	5	4
No distribution, trained on green balls	1	5	5	1	1	3
	2	5	5	2	1	1
	3	5	5	2	1	2
	4	5	5	1	1	3
	5	5	5	1	1	2
No distribution, trained on red balls	1	1	1	5	5	2
	2	1	1	5	5	2
	3	1	1	5	5	3
	4	3	2	5	5	3
	5	1	1	4	5	2
No Distribution, trained on blue tape	1	1	1	1	1	5
	2	1	1	1	1	5
	3	1	1	1	1	5
	4	1	1	1	1	3
	5	1	1	1	1	5

5.3.3 The Genetic Algorithm

Neural network architecture

The neural network has 120 input nodes, a single hidden layer consisting of 20 nodes, and an output layer with 5 nodes. The input nodes for the network come from the image received from the smart phone camera. The vision algorithm breaks the image into an 8×5 grid, and enhances each of the primary colors as described in Section 3.1.

The neural network's output layer has 5 nodes, representing the action that the robot should take. The five values correspond to the forward, left, right, reverse, and stop driving instructions for the robot. The genetic algorithm uses pressure to make losing values move far away from the winning value. The robot will always move in the direction of the winning value. The five output values are real numbers between -1 and 1, so while ties are possible, they are very rare and never occurred during this experiment.

The evolutionary algorithm

We used a standard generational genetic algorithm (GA) to evolve the neural network weights. Individuals are represented as a complete multilayer perceptron neural network; each individual has 2525 weights(120 input nodes to 20 hidden nodes = 2400, and 20 hidden nodes to 5 output nodes = 100 + 20 bias nodes). Initially the weights were randomly set in the range -1.0 to 1.0. Each generation after the first, and elitism of one is used.

Selection was a tournament of size three. Two individuals are selected as parents and produce only one offspring which receives crossover and mutation before being added to the next generation. Crossover was done as a uniform crossover on every set of parents when constructing the offspring. Each offspring will have 2525 weights which will be mutated at a rate of .2, on average resulting in 505 mutations per

Population size	40
Generations	300
Mutation rate	.2
Crossover rate	100% Uniform
Selection method	Tournament(size 3)

Table 5.3: Summary of the evolutionary algorithm parameters.

offspring. The amount of mutation was uniformly distributed between -0.1 to 0.1.

Fitness was determined by comparing the cases collected in the training data and the output that the trainer gave to the robot to the actions taken by each individual network with the same training case. The training cases are divided into category based expected action(Left, Right, Forward, Reverse, Stop). The network can earn an equal amount of fitness for getting all the cases in a category correct, regardless of the number of cases within the category. The fitness was calculated this way because during training, there were generally ten forward commands to every other command given. The algorithm for the fitness function can be seen in 2. The fitness function assigns a better fitness to the network if the network's output at the expected winning action is higher than it's output at it's other actions. The difference between the value corresponding to the expected action and the other actions was attempted to be wide, wide being defined as a threshold of 0.4. No testing was done on this threshold value, this value was chosen because it was a mid range value. High and low values were expected to work poorly in the fitness function. If the network chooses the correct action, it always gets a positive fitness for that training case, but the fitness will increment more if the if the difference is at least equal to the threshold.

The other parameters of the evolutionary algorithm are summarized in Table 5.3.

Distribution

While the generational genetic algorithm evolves, each generation the algorithm sends it's best individual to each of the other robots that it is evolving with. Each robot

Algorithm 2 Genetic Algorithm Fitness Function

```
for each network in population do
    fitness = 0
    for each training case do
        actualResults = network result from running training case
        expectedResults = results from training case
        winningValue = actualResults at the position of the expectedResults winner
        difference1 = winningValue - actualResults at first position other than winner
        difference2 = winningValue - actualResults at second position other than winner
        difference3 = winningValue - actualResults at third position other than winner
        difference4 = winningValue - actualResults at fourth position other than winner
        for each difference do
            if difference ≥ threshold then
                fitness += 0.5 / number of training cases in that catagory
            else
                fitness += 0.5 * (difference / threshold) / number of training cases in that catagory
            end if
        end for
    end for
    fitness = 1 - fitness / 5
end for
```

will receive the elites of every other robot each generation.

The robot puts the elites into a special portion of it's population. This portion of the population can be used for selection, but it is not overwritten with the next generation. This portion of the population is only overwritten by the elites sent to the robot from the other robots. This elite population is not given a fitness score, instead, 10% of the time, during selection the robot will select a random member of it's distributed population.

5.4 Results

First, our results show that parallel training does significantly reduce the total training time required. The scores received for the distributed demonstration case are clearly better than the scores received for the robot which was trained on all actions at the same time. This may be because simple demonstrations in a less noisy environment are easier to learn from.

Second, our results show that distributing behaviors, without distributing any

demonstrations do not result in behavior migration. Table 5.1 shows that distributed behavior cases behaved nearly identically to the robots trained only to do the same behaviors while not distributed.

Table 5.1 and Table 5.2 show that the robot was able to behave to avoid the blue tape even when the robot was not trained to do so. The fact that the non-distributed cases show this behavior as well leads us to believe that this in not a product of behavior migration, and is instead likely due to the way the robot learned to avoid the table. The neural network may have evolved to turn towards the white floors and walls.

5.5 Conclusion

Our results show that distributed LfD, using multiple robots, is a practical approach to reducing the time required to produce sufficiently rich training sets for learning. In addition, leveraging the multiple robots to create a distributed, island model, evolutionary algorithm will not, by itself, allow for behavior migration. We also show that collecting specific examples of behavior to create a demonstration set or distributed training, provides more robust behavior than a demonstration set create in a noisy environment.

This is a natural extension of the work done by Chernova and Veloso who combined training demonstrations collected in noisy environments. They pooled all the training data from multiple robots, and showed showed that they could reduce demonstration time. By distributing the demonstration process using the technique of training each robot on simple behaviors, demonstration time required can be reduced even further, for each trainer.

It's also important to note that even though we not show behavior migration in the distributed cases, our algorithm was not tuned to make behavior migration easy for the neural network. Specifically, other forms of crossover would probably work

better for migrating behavior data than a uniform crossover.

Although it was not done as an official experiment, these robots were trained tested on the problem of path following on two paths. The first path was indoors, with no walls to block the noisy input of the robotics lab. The robots were able to pick out the path and follow the path with zero errors. The second path was an outdoor trail where the robot traveled 400 meters (.25mi) through an arboretum.

Future work will include using learning from demonstration to navigate the robots in outdoor fields to seek out invasive species in lawns and agricultural fields.

Chapter 6: Conclusions

In this research, distributed evolution of neural networks has been shown to be effective in training robots to drive in multiple environments, and to complete individual and combined tasks. In particular, the results of the experiments in this thesis suggest that training robots on individual tasks, and distributing the training cases is more effective than training robots on a combined task, and distributing the behaviors as an island model during the learning process.

One persistent problem with the LfD model used in this research is the separation of the learning phase and the training phase. This is a problem because it does not allow a trainer to fix behavioral problems once the learning phase has finished. One goal for future work is to allow the robot to continue to learn and get feedback from operators during autonomous operation. This would allow the robot to get better at a task without needing extensive initial training sets, and thus reduce the need for a long training phase.

Preliminary work in this thesis shows that clicker training can be a useful tool in training robots to interact with their environment with training happening entirely in the field. This method of training allows the robot to receive real-time feedback from an operator, and could be combined with LfD to give the robot new training examples during autonomous operation.

References

[1] Ricardo Aler, Oscar Garcia, and José María Valls. Correcting and improving imitation models of humans for robosoccer agents. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2402–2409. IEEE, 2005.

[2] Ramesh Amit and Maja Matari. Learning movement sequences from demonstration. In *Development and Learning, 2002. Proceedings. The 2nd International Conference on*, pages 203–208. IEEE, 2002.

[3] Pierre Andry, Philippe Gaussier, Sorin Moga, Jean-Paul Banquet, and Jacqueline Nadel. Learning and communication via imitation: An autonomous robot perspective. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 31(5):431–442, 2001.

[4] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[5] Andrea Bauer, Dirk Wollherr, and Martin Buss. Human–robot collaboration: a survey. *International Journal of Humanoid Robotics*, 5(01):47–66, 2008.

[6] Andreas Birk and Stefano Carpin. Rescue roboticsa crucial milestone on the road to autonomous systems. *Advanced Robotics*, 20(5):595–605, 2006.

[7] Josh C Bongard. Accelerating self-modeling in cooperative robot teams. *Evolutionary Computation, IEEE Transactions on*, 13(2):321–332, 2009.

[8] Cynthia Breazeal, Daphna Buchsbaum, Jesse Gray, David Gatenby, and Bruce Blumberg. Learning from and about others: Towards using imitation to bootstrap the social understanding of others by robots. *Artificial Life*, 11(1-2):31–62, 2005.

[9] Sylvain Calinon, Eric L Sauser, AG Billard, and DG Caldwell. A learning by imitation model handling multiple constraints and motion alternatives. In *Intl Conf. on Cognitive Systems (CogSys)*, 2010.

[10] Genci Capi and Hideki Toda. Evolution of neural controllers for robot navigation in human environments. *Journal of Computer Science*, 6(8):837, 2010.

[11] Sonia Chernova and Manuela Veloso. Teaching collaborative multi-robot tasks through demonstration. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 385–390. IEEE, 2008.

[12] Sonia Chernova and Manuela Veloso. Confidence-based multi-robot learning from demonstration. *International Journal of Social Robotics*, 2(2):195–215, 2010.

[13] J Clouse. Phd: On integrating apprentice learning and reinforcement learning title2. 1997.

[14] Sevan G Ficici, Richard A Watson, and Jordan B Pollack. Embodied evolution: A response to challenges in evolutionary robotics. In *Proceedings of the eighth European workshop on learning robots*, pages 14–22. Citeseer, 1999.

[15] Evert Haasdijk, Arif Atta-ul Qayyum, and Agoston Endre Eiben. Racing to improve on-line, on-board evolutionary robotics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 187–194. ACM, 2011.

[16] Evert Haasdijk, AE Eiben, and Giorgos Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE, 2010.

[17] Robert-Jan Huijsman, Evert Haasdijk, and AE Eiben. An on-line on-board distributed algorithm for evolutionary robotics. In *Artificial Evolution*, pages 73–84. Springer, 2012.

[18] Cynthia L Johnson and Avelino J Gonzalez. Learning collaborative team behavior from observation. *Expert Systems with Applications*, 41(5):2316–2328, 2014.

[19] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[20] Giorgos Karafotias, Evert Haasdijk, Ágoston E Eiben, Evert Haasdijk, A Eiben, A Winfield, Evert Haasdijk, A Rusu, A Eiben, A Eiben, et al. An algorithm for distributed on-line, on-board evolutionary robotics. In *GECCO*, pages 171–178, 2011.

[21] Sean Luke and Vittorio Amos Ziparo. Learn to behave! rapid training of behavior automata. In *and Learning Agents Workshop at AAMAS 2010*. Citeseer, 2010.

[22] Luis Nunes and Eugénio Oliveira. Advice-exchange in heterogeneous groups of learning agents. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 1084–1085. ACM, 2003.

[23] Luís Nunes and Eugénio Oliveira. Cooperative learning using advice exchange. In *Adaptive agents and multi-agent systems*, pages 33–48. Springer, 2003.

[24] Luís Nunes and Eugénio Oliveira. Advice-exchange between evolutionary algorithms and reinforcement learning agents: Experiments in the pursuit domain. In *Adaptive Agents and Multi-Agent Systems II*, pages 185–204. Springer, 2005.

[25] Eugnio Oliveira and Luis Nunes. Learning by exchanging advice. *STUDIES IN FUZZINESS AND SOFT COMPUTING*, 162:279–314, 2004.

[26] Stuart Russell. *Artificial intelligence: A modern approach, 3/E.* Pearson Education India, 3rd edition, 2010.

[27] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[28] David Silver, J Andrew Bagnell, and Anthony Stentz. Applied imitation learning for autonomous navigation in complex natural terrain. In *Field and Service Robotics*, pages 249–259. Springer, 2010.

[29] David Silver, J Andrew Bagnell, and Anthony Stentz. Perceptual interpretation for autonomous navigation through dynamic imitation learning. In *Robotics Research*, pages 433–449. Springer, 2011.

[30] Terence Soule and Robert B Heckendorn. Cotsbots: computationally powerful, low-cost robots for computer science curriculums. *Journal of Computing Sciences in Colleges*, 27(1):180–187, 2011.

[31] Keith Sullivan and Sean Luke. Multiagent supervised training with agent hierarchies and manual behavior decomposition. In *Proceedings of Agents Learning Interactively from Human Teachers Workshop*, 2011.

[32] Keith Sullivan, Sean Luke, and Vittoria Amos Ziparo. Hierarchical learning from demonstration on humanoid robots. In *Proceedings of Humanoid Robots Learning from Human Interaction Workshop*, 2010.

[33] Dejvuth Suwimonteerabuth and Prabhas Chongstitvatana. Online robot learning by reward and punishment for a mobile robot. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 921–926. IEEE, 2002.

[34] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, volume 337. Amherst, MA, 1993.

[35] Richard A Watson, Sevan G Ficici, and Jordan B Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 2002.

[36] Steven D Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *AAAI*, pages 607–613, 1991.

[37] Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Sam Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14(2-3):225–237, 2003.