On-line, On-board, Evolutionary Learning from Demonstration in Autonomous, Mobile Robots

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

Nathaniel Ebel

May 2014

Major Professor: Terence Soule, Ph.D.

Authorization to Submit Thesis

This thesis of Nathaniel Ebel, submitted for the degree of Master of Science with a major in Computer Science and titled "On-line, On-board, Evolutionary Learning from Demonstration in Autonomous, Mobile Robots," 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	______________________________ Terence Soule, Ph.D.	Date____________
Committee Members	______________________________ Robert Heckendorn, Ph.D.	Date____________
	______________________________ Barrie Robison, Ph.D.	Date____________
Department Administrator	______________________________ Gregory Donohoe, Ph.D.	Date____________
Discipline's College Dean	______________________________ Larry Stauffer, Ph.D.	Date____________

Final Approval and Acceptance by the College of Graduate Studies

	______________________________ Jie Chen,, Ph.D.	Date____________

Abstract

The ability to easily train, and interact with robotic agents has been the focus of a great deal of research within the fields of Artificial Intelligence and Machine Learning. Learning from Demonstration (LfD) is a common approach for teaching robotic agents new behaviors because it requires little specialized knowledge of robotic hardware or of computer programming. Evolutionary learning techniques have also commonly been applied to robotic learning tasks, but the two techniques have rarely been used together due to the computational requirements of running an evolutionary process on-line, and on-board a robot during the LfD process. This thesis presents an on-line, on-board, evolutionary learning from demonstration protocol that enables autonomous robotic agents to learn how to complete a path following task. In addition, the robots are capable of recording their GPS location during autonomous operation which has a wide variety of practical applications from autonomous trail mapping, to marking objects of interest in disaster response applications.

Acknowledgements

I would like to thank Dr. Soule for all the guidance and advice throughout my time in the graduate program at the University of Idaho and for acting as my Major Professor.

I want to thank Dr. Soule and Dr. Heckendorn for inviting me into the research lab, and supporting our work with their time and expertise.

I thank Dr. Heckendorn for acting as a committee member and for always bringing an interesting perspective and a wealth of knowledge to our work.

I would like to say thank you to Dr. Robison for agreeing to serve as the outside member of my committee and for providing an interesting outside perspective to the work.

I want to thank Travis DeVault, Juan Marulanda, and Jayandra Pokharel for all of the assistance, advice, friendship, and laughs during my time in the program. Enjoying the people you work with is priceless. I also want to thank Juan again for going out of his way to talk with me about my research interests on my first day of classes and for pointing me in the direction of Dr. Soule and Dr.
Heckendorn.

I would like to thank my family for always being supportive, encouraging me to apply for grad school, and for helping teach me how to enjoy learning.

Last, but not least, I want to thank my wife for postponing her dream of going to grad school and for always supporting me in everything I do.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In recent years, there has been an increase in interest in the field of evolutionary robotics. This research focuses on applying evolutionary methods and algorithms towards the creation of robotic controllers [14]. Robots that can evolve and adapt autonomously are desired because of the difficulty in hand coding or simulating controllers that can handle complex and dynamic environments [37]. If a robotic agent can continuously adapt and evolve in real time then it is capable of operating within a variety of environments without needing to modify its controller. Examples of such environments include natural disaster areas and extraterrestrial landscapes, both of which are prime candidates for utilizing autonomous mobile robots [8, 61].

Much of the work in evolutionary robotics has focused on optimizing a controller to allow a single robot to complete a given task. The most common approach is off-line evolution in which the robotic agent is evolved and evaluated in simulation, and the best individual solution is transferred to a physical robot [32]. While this approach works well for some problems, it has several important limitations. Firstly, controllers evolved off-line cannot adapt to dynamic environments. Secondly, it is difficult to simulate real world environments, particularly when dealing with inputs such as lighting, texture, and sound [14]. Because of this, controllers created during simulation are susceptible to decreases in effectiveness once transferred to the real world [11, 35]. To create controllers that perform well in real-world environments it's necessary for controllers to be tested in the physical world during embodied trials.

Embodied trials refers to the testing of an individual's fitness on a physical robot in the real world so real inputs are received. This approach gives more accurate fitness evaluations, but evolution is still done off-line on a remote machine [45]. This means that evolution is dependent on some third party system and the physical agent is not truly autonomous. Embodied trials allow a robot to adapt to its environment as it operates, but the robot must wait for evolution to take place on the external machine. If this process has a large enough latency, the robot's environment could potentially have significantly changed, causing the new controller to be ineffective. Off-board evolution with embodied trials is also dependent on the connection of a robot to the machine carrying out the evolutionary process. If this connection is broken, the robot can no longer evolve, potentially leaving the robot with an ineffective controller. This drawback limits the effectiveness of the embodied, off-board approach in real world applications and leads to a need for robots that can continuously evolve in real time in real world environments.

Modern approaches for evolutionary robotics are moving towards fully embodied evolution where both fitness trials and evolution are done on-board the physical robot [18]. In this type of system, the robot is continually evolving and testing its fitness with real-world inputs using only the hardware available on the robot itself. This gives the robot a level of automation and adaptability not possible with off-line, off-board approaches.

Learning from demonstration (LfD) is a robotic training method in which a robotic agent learns from observing demonstrated behaviors. The robot is able to learn to replicate observed behavior to operate autonomously and perform a desired task. Because demonstrating an action or behavior to a robot typically takes no special knowledge of programming or robotic hardware, LfD is a promising approach for increasing the adoption and effectiveness of human-robot interactions.

This thesis presents an on-line, on-board evolutionary Learning from Demonstration system that can be implemented on a low-cost, but computationally powerful, mobile robotic platform. This allows a human operator with no specialized knowledge of programming or robotic hardware to train a mobile robot to complete some given task. The robot learns by observing the actions demonstrated by a human operator and then evolving a neurocontroller to replicate this behavior. This system has been tested and shown to be capable of training a mobile robot to complete path following tasks in both indoor and outdoor environments. In conjunction with development of the on-line, on-board evolutionary LfD system, work was done to develop robotic vision systems to increase the image processing capabilities of the robotic platform and to develop a method of tracking the robots position during autonomous operation. This additional work increases the utility of real world applications of the proposed robotic system.

The rest of this document is organized as follows: Chapter 2 gives an extensive background section for evolutionary computation, evolutionary robotics, and learning from demonstration, Chapters 3 and 4 were both written as stand alone papers and describe two sets of experiments using the proposed on-line, on-board evolutionary Learning from Demonstration system, Chapter 5 describes some of the technical challenges that were addressed during the course of this work, Chapter 6 gives conclusions drawn from this work, and Chapter 7 describes several avenues for future work.

Chapter 2: Background

2.1 Evolutionary Computation

Evolutionary computation is a sub field of computer science in which the fundamental principle of biological evolution, survival of the fittest, is applied to a wide range of optimization and design problems in a variety of fields including satellite beam design [33], molecular sequence alignment [43, 62], and control of competitive video game agents [21]. A great deal of research has been devoted to evolutionary computation, much of which focuses on how to emulate evolution to solve a particular problem. This typically involves answering several questions related to algorithm design [24]:

- How should genetic material be organized?
- How will selection take place?
- How will populations be structured?
- What will reproductive constraints look like?
- Where and how will mutations occur?

2.1.1 Representation

How an individual should be represented is often one of the most difficult considerations when designing an evolutionary algorithm. It is often referred to as the "Representation Problem" and can have a great impact on the fitness on an individual. A potential solution to a given problem is the phenotype of an individual while its genetic encoding is its genotype [20]. It's important to choose a representation that can allow good alleles, pieces of genetic code, to stick together [24]. Within evolutionary robotics, a common genotype representation is to treat the weights of an artificial neural network (ANN) as individual genes within the genotype encoding.

2.1.2 Selection

Parent selection works to improve the fitness of a population by choosing individuals with high fitness to pass on their genetic information to new generations [20]. This is typically a stochastic process in which the probability of being selected for reproduction is proportional to an individual's fitness leading to a greater chance of highly fit individuals reproducing. Tournament selection is one approach in which a set of random individuals are selected, and the most fit of those individuals is chosen to reproduce [24]. While the goal of selection is to improve the fitness of the population, the most fit individual isn't always chosen. The small chance for a poorly fit individual to reproduce allows an algorithm to avoid getting stuck in a local optimum. Retaining diversity in the population can lead to better exploration of the search space and may lead to a better performing algorithm.

2.1.3 Population Structure

The population of an evolutionary algorithm is a collection of genotypes representing possible solutions to a particular problem. Evolution works at the population level where individual genotypes don't change or evolve, but the population's overall fitness can increase through reproduction [20]. Defining the population for an algorithm is usually as simple as defining the population size. The size of every generation's population will typically remain constant but can change with each new generation. The size of the population must be 1 or larger, but the efficiency of the evolutionary process should be considered when choosing how large the population should be. Larger populations mean more individuals must be evaluated and the time between generations will increase which may decrease the effectiveness of the algorithm.

2.1.4 Reproduction

Reproduction allows for new generations of individuals to be created and improve the population [20]. Genetic information is taken from a single or multiple parent individuals to create a new offspring individual. The exchange of genetic information takes place during crossover between parents. This can take the form of one point, two point, or uniform crossover [19].

- One point - picks a point in the genotype and all genes after this point are swapped between genotypes
- Two point - picks two points, and the genes in between them are exchanged.
- Uniform - All genes have a probability of being exchanged between parents

Once crossover has been applied and new individuals have been created mutation can occur to introduce new genetic material into the genotype. Mutation typically has a constant probability to modify each individual gene in the genotype.

2.1.5 Different Flavors of Evolutionary Computation

Implementations of evolutionary approaches can be grouped into several main subtypes:

- Evolutionary Algorithms (EAs)
  - Genetic Algorithms (GAs)
  - Evolutionary Programming (EP)
  - Evolution Strategies (ES)
- Genetic Programming (GP)

Evolutionary algorithms work to optimize the parameters for a problem while genetic programming focuses on building descriptions of objects that carry out computations

[19]. GAs, EP, and ES all work to optimize parameters, but go about it in different ways. GAs are the most common type of evolutionary algorithm and represent genotypes as strings of varying lengths and/or types. EP works to evolve adaptive behaviors by modifying parameters of some fixed control structure. ES are usually applied to optimization problems and have an emphasis on using mutation to create new genotypes [20]. Genetic programming has two key distinctions from the previously described algorithms. The goal of GP is to evolve a description of some object or controller than can carry out a task rather than optimising a given control structure as in EAs [20, 19]. GP also usually represents its solutions as trees rather than strings.

2.2 Evolutionary Robotics

These key concepts of evolutionary computation play an integral role within evolutionary robotics. Population structure and the method reproduction are particularly important when evolving robotic controllers on-line and on-board. In [10] its stated that because of limited processing power in mobile robots, highly efficient evolutionary algorithms with small population sizes are needed to be effective at evolving robotic controllers. With the steady increase of general processing power, and the rapid increase in prevalence of powerful smartphone and tablet devices, processing power isn't nearly as limited as it used to be. However, carefully designing efficient algorithms with appropriate population sizes can significantly increase the performance of on-line on-board evolution, and it may still be necessary to utilize simple algorithms with small population size depending on the robotic platform used.

2.2.1 Classification

To effectively discuss different implementations of evolutionary robotics it is helpful to make use of a general classification scheme. Several classification systems have been proposed in previous works [59, 19]. In [59], a classification of evolutionary robotics was described in which the embodiedness of a robot was the crucial classifier. This classification was expanded in [19] and classified embodied evolution based on where the evolutionary processes take place. In this system it is assumed that the robot is embodied because it will operate in the real world, and therefore removes non-embodied evolution from the classification all together.

Along with a new method of classification, [19] presents a vocabulary with which to effectively describe evolutionary robotics. The term “embodied” refers to the physical testing of fitness within a real-world, physical robot. The rest of the vocabulary focuses on the when, where, and how of the evolutionary process.

1. “off-line” (evolution occurs before run time)
2. “on-line” (evolution occurs during run time)
3. “off-board” or “extrinsic” (evolution not performed on the robot)
4. “on-board” or “intrinsic” (evolution is performed on the robot)
5. “encapsulated” or “centralised” (evolutionary process is restricted to a single robot)
6. “distributed” (evolution is performed across multiple robots)

2.2.2 When

When evolution occurs has a large impact on how the robot can evolve. If evolution stops when the robot is deployed then it’s using off-line evolution and it can no longer adapt to its environment [19]. This requires the robot’s controller to be effective for a particular task at the moment of deployment. On-line evolution allows the evolutionary process to continue for as long as the robot is operational.

2.2.3 Where

Evolution can take place both on-board (intrinsic) or off-board (extrinsic) the robot. In the on-board case, evolutionary operators such as crossover and mutation are applied within the robot itself. This contrasts with the off-board approach in which evolution is controlled by a remote machine that performs the evolutionary operations, and loads newly created genetic information onto the robot. Historically, the off-board approach has been much more common due to the limited processing power of robotic hardware.

2.2.4 How

The "How" in [19]'s classification refers to how evolution is managed for either a single or multiple robots. In an "encapsulated" or "centralised" approach a single robot undergoes self contained evolution. The robot maintains a population of genotypes and uses a time sharing system to take turns activating and evaluating individual genotypes for evaluation. The overall fitness of a robot is the average fitness of each individual in the on-board population. The "distributed" approach uses a group of robots to emulate real world evolution. Each robot contains one genotype and is controlled by the associated phenotype. Robots can share genetic information by broadcasting genetic information to nearby robots, and can then use this shared information to reproduce. In this case, each robot will have its own fitness taken from its single on-board individual, but the overall fitness of the population is still calculated as the average of each individual in the total, distributed population.

With this classification it's possible to clearly define different approaches for evolving robotic controllers. It's with this classification system and vocabulary that the rest of this paper will be described with a focus on on-line, on-board evolutionary approaches.

2.3 Robotic Learning

The ability to learn new behaviors is very desirable in autonomous robots as it allows a single robot to be applied to a variety of situations and environments. This has several key benefits:

1. Development time and money are saved when a robot needs to be deployed in a new domain

2. If a robot's objectives change during operation, its controller doesn't have to be modified

These benefits allow robots to work in a variety of environments towards the completion of tasks with varying complexity. This flexibility eliminates the need for custom development on a per-task basic which saves money, and can save crucial time in time sensitive situations such as natural disaster response.

2.3.1 Types of Learning

Robotic learning can be generalized into two broad categories: *unsupervised* and *supervised*. Unsupervised learning includes forms of learning where no external mechanism is giving feedback to the learning system. An example of this is principal component analysis where similar data is clustered without an external input as to how the data is related. In evolutionary robotics, unsupervised learning takes place when a robot is responsible for calculating its own fitness as a measurement of its performance at a given task. Supervised learning systems generally involve an external source providing examples of desired behavior to the robot, and the robot learning to mimic, and generalize those behaviors. Back propagation to train a neural network is an example of supervised learning. As a part of this thesis, another form of supervised learning was used: Learning from Demonstration.

Learning from Demonstration

Learning from Demonstration involves the demonstration of desired behaviors to a robotic agent so that agent can then learn what actions to take given a particular set of inputs. This is similar to how people often learn a new task from someone who is more experienced and can show the novice how to react in a given situation. Learning from Demonstration has been successfully applied to autonomous robots in a variety of fields ranging from the learning of soccer behaviors [55], to the training of a large autonomous vehicle to drive through outdoor terrain [46]. Chapter 3 presents an evolutionary learning from demonstration system for autonomous mobile robots.

2.4 Review of Related Work

2.4.1 Encapsulated Approaches

Encapsulated evolution is the most common approach for on-line, on-board evolution. In this system each robot runs its own evolutionary algorithm on-board, maintaining a population of potential controllers, and testing each of those controllers on-line in a time sharing system. The performance of the robot is then calculated as the average performance of each controller in the population as the performance of one controller can impact the performance of another.

(1+1) ON-LINE

One of the first methods of encapsulated evolution is (1+1) ON-LINE which is based on the (1+1) evolutionary algorithm and was proposed in [10]. In this system a population of size 1 is kept to maintain the "champion" individual. The champion is the genotype with the highest fitness so far. In each generation, a single offspring is generated by applying a mutation operator to the champion. This individual is then tested on-line by activating it as the robot's controller and letting it control the robot during some test task. After a maximum number of time steps, T_{max}, if the fitness of the current controlling genotype is greater than that of the champion, the champion is replaced. It's through this process that the overall performance of the robot is increased.

During evolution, mutation occurs with a set probability, $P_{mutation}$, and a mutation step size of σ. To balance the "exploration" and the "exploitation" of the algorithm this σ value can change during runtime to change between global and local search. When a new genotype causes an increase in performance the step value is decreased to exploit the local search space of the individual. Once improvement from a local search slows, the step size is increased again to perform a more global search and break away from a potential local minima.

One challenge with on-line, on-board evolution is the effect that a previous genotype can have on the fitness of its successor. Because genotypes are continuously generated, activated, and tested, a new controller has to take over in the same state the previous one left the robot in. If the the previous controller had a poor fitness it may have left the robot in a particularly difficult situation that places the next activated controller at a disadvantage. This could punish an otherwise fit individual and lead to throwing away a useful genotype. To reduce the effect of a previous controller on a newly activated one, the idea of a "recovery time" has been successfully applied in [10, 19]. The recovery time is a set period of time right after a controller is activated in which fitness is not tested. This allows the new controller time to gain control of the robot and move it out of any low fitness state. After the recovery time is up, fitness testing begins. This process serves to more fairly evaluate controllers against one another.

Fitness evaluations in the real world are inherently noisy for a variety of reasons including complex lighting and sounds, shadow, texture, and imperfect sensing equipment. These factors, coupled with controllers that can be impacted by their predecessors, make it important to question whether a particular "champion" individual is really a strong solution or if it got lucky during its evaluation. To verify that a champion controller is, in fact, a strong solution reevaluation is commonly applied [10, 19]. With some probability, $P_{reevaluate}$, the champion will be activated and retested. If the new fitness value is poor, it will be replaced by an individual from the "hall of fame" which is a list of previously well performing individuals.

$(\mu + 1)$ ON-LINE

The (1+1)ON-LINE algorithm was extended to $(\mu+1)$ ON-LINE in [10]. This approach was refined and presented further in [19, 26]. The $(\mu + 1)$ ON-LINE approach is based on classic $(\mu + 1)$ evolution strategies because of their proven success as numerical optimisation tools. Rather then a population size of 1 as in $(1 + 1)$ ON-LINE, $(\mu + 1)$ ON-LINE maintains a population size of μ and either generates a single, new individual or reevaluates an existing individual at each time step.

$(\mu+1)$ ON-LINE attempts to handle noisy fitness calculations be reevaluating individuals in the population with a probability, $P_reevaluate$. This ensures that an individual's high fitness wasn't a result of luck. Binary tournament selection is used to choose an individual for revaluation, and its fitness is updated using an exponential moving average. This process works to increase the overall fitness of the population by ensuring individuals perform well within a broad range of operating states.

If an individual is not being reevaluated, then a new individual is generated using binary tournament parent selection, an averaging crossover, and a mutation step. In [26], it was found that the standard evolution strategies approach of self-adapting σ (mutation step) values produced better results when compared against both a low(0.2) and a high(0.8) fixed value. Once a new individual is generated, it undergoes the same recovery period that was used in (1+1) ON-LINE, reducing the effect the previous controller has on the fitness of the new individual. When the recovery period is over, the individual's fitness is tested and calculated. If the new individual's fitness is greater than the worst individual in the population, the new individual will replace the old one in an effort to increase the overall performance of the population. The process of replacing old individuals with new ones comes with the risk of a new, high performing, individual being lucky in its testing conditions and replacing a generally more fit existing individual. That is why reevaluation is a key component in $(\mu + 1)$ ON-LINE.

The effectiveness of $(\mu + 1)$ ON-LINE at generating robotic controllers, and the effect of different algorithmic parameters on performance was tested in [10, 19, 26]. Testing was done using simulated e-puck robots within the Webots[a] simulation environment. Tests showed the algorithm was successful at evolving robotic controllers, and that σ (mutation size) was the most influential parameter on performance. Large population sizes and high reevaluation rates lead to an overall decrease in performance with the smallest tested values of 6 and 0.2 respectively producing the best results. It was also noted that fitness evaluation time was the second most influential parameter and longer evaluation times lead to more stable populations which are desirable because of the inherent time sharing aspect of embodied, encapsulated approaches.

(1 + 1) ON-LINE with Restart

The (1+1) ON-LINE approach uses an adaptive σ value to balance between global and local search. This allows the algorithm to explore the global space while exploiting good solutions. The work in [41] points out that while exploring the global space, (1+1) ON-LINE only explores regions with higher fitness than the current champion. Because of this, the probability of finding a more fit global optima is decreased. A more preferable search behavior would be to explore a less fit region that slowly leads to a global optimum through local search.

Rather than modify the mutation step size to adjust this search behavior, [41] proposed the (1+1)-Restart-Online algorithm in which the algorithm's search is sometimes

[a]Webots http://www.cyberbotics.com

restarted from a random individual in order to more fully explore the search space. The algorithm restarts when the search has "stalled" which is decided by the number of times an individual has been reevaluated. For the experiments presented in [41], a Reevaluation$_{max}$ threshold of seven was set. When the number of reevaluations reaches Reevaluation$_{max}$ the current champion is saved to the Hall-of-Fame and new random champion and challenger genotypes are created to restart the search.

The performance of (1+1) ON-LINE and (1+1)-Restart-Online was compared by examining the Hall-of-Fames generated during simulated experimentation. It was determined that including restart in the algorithm increased the speed at which fit individuals were found, and more of the global search space was explored.

(μ + 1) ON-LINE with Racing

One of the key limiting factors in on-line, on-board evolution is the small number of phenotype evaluations that occur within a reasonable time for real world robotics. In [27], this limitation was addressed by applying racing to the ($\mu + 1$) ON-LINE algorithm. Racing is a process in which models are discarded as soon as they are determined to be a poor choice. When applied to the (1+1) ON-LINE algorithm it means discarding an individual if its intermediate fitness indicates it will not be a highly performing individual with the goal of maximizing the time spent evaluating more promising individuals.

To apply racing effectively, two criteria must be met: intermediate fitness must be calculable, and intermediate fitness must be comparable to final fitness. Intermediate results are often generated by evaluating the current fitness at an intermediate time and then averaging that fitness over the time steps that have elapsed. This allows it to be compared to the final fitness of the population.

In (μ + 1) ON-LINE with Racing, at an intermediate time between T_0 and T_{max}, fitness is calculated. If the intermediate fitness suggests the individual will turn out worse than the current worst in the population, it's considered a waste of time to continue evaluation. To determine if an individual is likely to be worse than the worst in the current population, the following version of the Hoeffding inequality is used:

1. $F_{challenger} \geq F_{worst} - 2\xi(t)$

   with $\xi(t)$ calculated as:

2. $\xi(t) = \sqrt{\frac{(F_a - F_b)^2 \log(2/\alpha)}{\beta t}}$

   - F_a = the best individual
   - F_b = the worst individual
   - α = the significance level
   - β = parameter to tune comparison strictness

With these equations is is more likely that an individual will continue to be evaluated early in the evaluation process, and the chance to continue evaluation decreases over time.

The effectiveness of adding racing to the $(\mu + 1)$ ON-LINE approach was tested using several experimental problems. It was shown that $(\mu + 1)$ ON-LINE with Racing significantly increased performance for all tasks. This was true regardless of the parameters used in the Hoeffding equations. The results suggest that racing can be an effective way of increasing the overall performance of a population of robot controllers.

Controlling evaluation of $(\mu + 1)$ ON-LINE

(1+1) ON-LINE, $(\mu + 1)$ ON-LINE, and their variants have shown success in evolving controllers during on-line operation. However, these approaches still use a number of pre-tuned parameters. For best performance it is desirable to allow parameters to be tuned during on-line operation so controllers may adapt to the current environment and task. The length of evaluation time has a significant impact on the performance of each of the encapsulated approaches described previously. [6] examines an adaptive approach to tuning evaluation time.

Parameter modifications fall into one of two categories: Parameter Tuning, and Parameter Control. Parameter Tuning is the choosing of parameter values before operation by testing different values and choosing the best one. Parameter Control allows the evolutionary process to modify parameter values. There are several methods of Parameter Control including: deterministic, adaptive, and self-adaptive control. The work in [6] uses an adaptive control approach in which the max evaluation time (T_{max}) is modified based upon the performance of the evolutionary algorithm. Two different adaptive control methods were tested: Roulette Wheel Selection (RWS), and Heuristics-Rule (H-Rule).

The RWS method uses a range from 1 to 1000 as acceptable values for T_{max}. This range is initially split into equal regions, and then a region is randomly selected. From the selected interval, a T_{max} value is selected from a uniform distribution. When an individual is evaluated, if its fitness is greater than the last individual's fitness the selected interval's selection weight is increased by 1, otherwise if the fitness is less than the previous individual's the weight is decreased. An interval's selection weight can never be less than 1, so it always has some chance of being selected. This process leads to T_{max} intervals of high performing individuals being selected more often.

The H-Rule method modifies T_{max} if 3 individual evaluations in a row have not lead to an increase in performance. If no increase occurs, the algorithm randomly adds or subtracts one third of the current T_{max}.

These two Parameter Control methods were used to control the T_{max} parameter of the $(\mu + 1)$ ON-LINE with Racing. Experiments show that these adaptive approaches work well enough to eliminate the need to pre-tune the T_{max} parameter. The Parameter Control methods have average fitness similar to that of pre-tuned parameter values. The RWS method performs as well as the best pre-tuned parameter values, and performs slightly better in dynamic tasks in which the goal changes during operation. Performance of the H-Rule method is slightly less than the best pre-tuned values, but does outperform pre-tuned parameters in dynamic tasks. The RWS method was also applied to the racing parameters of α and β to allow for dynamic control of the Racing algorithm. This removes the need to pre-define these parameters as well.

Early Example of Evolution in a Real Robot

In [23], Floreano and Mondada demonstrated an on-board approach to the evolution of neural controllers for autonomous agents. A key point was to demonstrate that performing evolution in real hardware is possible using modern hardware.

They employed a standard genetic algorithm using fitness scaling, roulette wheel parent selection, a standard mutation operator, and one-point crossover. Evolution maintained a population size of 80 individuals, ran for 100 generations, and each evolved controller was tested for 80 actions before fitness was assigned. Fitness was calculated as a function of the average wheel speed, the difference between the wheel speeds, and the activation level of the closest distance sensor. The controlling neural network included a single input layer of eight sensor nodes, and two output nodes to control the robot's two motors. The robotic platform used was the Khepra because of its small size and simple, uniform design.

The Khepra was evolved to perform an obstacle avoidance task. Results showed that evolution was capable of producing controllers to complete the task in less then the allotted 100 generations. Typically, optimal behaviors were present after 50 generations. It was noted that several emergent behaviors were present including the regulation of speed and driving in the forward direction. This showed evolution was capable of producing complex solutions to a simple problem.

rtNEAT

rtNeat was introduced in [50] as a real time variation of NeuroEvolution of Augmenting topologies (NEAT). rtNEAT was originally used to evolve virtual agents within a game developed as part of the NeuroEvolving Robotic Operatives (NERO) project. It was applied to robotics in [45] as a means of evolving robotic controllers, and was compared to odNEAT, a distributed variation of NEAT.

To run NEAT in real time, individuals must be generated and replaced continuously rather then at the end of a generation. This ensures that observed behaviors don't suffer from large, sudden changes. Because individuals in the population may have different network topologies, the term species is used to describe groups of individuals with the same topology. During each time step the algorithm performs the following actions:

1. Remove the matured individual from the population that has the lowest adjusted fitness
2. Re-estimate the adjusted fitness for each species in the population
3. Select a parent species to create a new offspring
4. Adjust compatibility threshold and reassign all individuals to a species. This ensures there are never too many or too few species.
5. Place the new individual in control with the environment

rtNEAT was tested in [45] on a group aggregation task. The experiment consisted of 30 trials using groups of 5 robots (simulated e-pucks). Two changes were made to rtNEAT for the purpose of these experiments so the results could be compared to those of odNEAT (described in 2.4.2). First, the compatibility threshold was static throughout the lifetime of the algorithm. Secondly, rather than creating an offspring at set time intervals, offspring were created when a robot's virtual energy level reached zero. A population size of 200 individuals was maintained and distributed over the five robots so that each robot maintained an on-board population of 40 individuals. Results of testing showed that rtNEAT could produce effective controllers for the task of group aggregation. The average evolution time was 4.44 ± 3.27 simulated hours. This was faster than the 6.22 ± 5.55 simulated hours that odNEAT took to evolve a solution.

2.4.2 Distributed Approaches

Distributed evolution spreads the evolutionary process throughout a group of robots each of which typically has a single genotype stored in its memory. Since each robot represents an individual genotype, the group of robots as a whole constitute the evolutionary population. Evolutionary operations such as reproduction occur through robot interactions in which genetic information is passed between robots. This closely imitates real world group interaction and reproduction. Distributed evolution produces individual genotypes, but the overall fitness is dependent on the fitness of the population as a whole. Because there are multiple robots working independently, evolution occurs concurrently, and may lead to a wider exploration of the potential solution space.

Early work of Watson et. al

The work of Watson et. al in [22, 60, 59] laid the groundwork for much of the subsequent embodied evolution research. They worked to solve several of the key problems in evolutionary robotics through a distributed, embodied approach. Firstly, by using an embodied approach with real world fitness evaluations, the sim-to-reality gap was eliminated. Secondly, by distributing evolution across a population of robots, the lengthy evaluation time of time-sharing systems was avoided.

The distributed approach stored a single genotype in each individual robot. The robots operated autonomously maintaining a virtual energy level that represents a robot's fitness. As robot performance increases, so does its energy level, and when a robot attempts to reproduce its energy level is decreased. A unique feature of this approach to fitness evaluation is that the energy level was not reset when a new genotype became active, so the energy level was a reflection of the performance of all genotypes that had been active within an individual robot.

No individual learning occurs within a robot, so improvements in the population must come from robot interactions and reproduction. A probabilistic implementation of the Microbial GA was used evolve the population of robots. The Probabilistic Gene Transfer Algorithm (PGTA) is a probabilistic variant of the Microbial GA in which 2 individuals are selected and genes from the less fit individual are replaced by genes from the more fit individual to effectively create an offspring. In PGTA, a gene from an individual is mutated and broadcast over local communication probabilistically at a rate proportional to the individual's current energy level. If the broadcasted gene is received by another robot, it may probabilistically be accepted into its genotype at a rate inversely proportional to is energy level. Through this process, highly fit individuals attempt to spread their genes while resisting the incorporation of new genes from other individuals.

This evolutionary approach was implemented and tested on actual robotic hardware using evolved neural networks to control the group of 8 robots towards the completion of a simple phototaxis task. Results showed that the distributed, embodied approach outperformed hand coded controllers, and demonstrated the first distributed, embodied evolutionary approach.

PEGA

Like the previously described work of Watson et. al [22, 60, 59], [42] demonstrated a distributed, embodied approach referred to as the Physically Embedded Genetic Algorithm (PEGA). It was quite similar to previous approaches, but it transferred a genotype as well as its fitness to other individuals.

PEGA utilized a (1+1) population structure in each individual robot within the population. A currently active genotype was kept and used to control the behavior of the robot, while a "champion" genotype was kept and used to revert to when the active genotype's fitness was worse than that of the champion. For the experiments done in [42] two physical robots were used with a communication range of one meter.

During operation, each robot would execute behaviors based on their currently active genotype and evaluate its fitness for a fixed amount of time. After the allotted time was up, the robots would move within communication range of one another to exchange their current genotype and associated fitness.

When information was exchanged, crossover or mutation were probabilistically applied to the received genotypes. Crossover occurred in two situations. First, if a received genotype was more fit than the robot's active genotype half of the active genotype was replaced with the corresponding half from the received genotype. Second, if the received genotype was less fit than the active genotype, but the robot's champion genotype had a better fitness, then there was a 30% chance that a random bit of the active genome would be replaced by the corresponding bit from the champion. The other 70% of the time no crossover was applied, and instead mutation was probabilistically applied to the active genome based on its fitness.

The PEGA approach was tested on several tasks including phototaxis, obstacle avoidance, and robot seeking. Experiments were done to evolve controllers that could solve a single task and could solve multiple tasks based on the current environment.

The results of experimentation showed that PEGA was capable of evolving controllers to solve the simple test tasks, and that more complex controllers could also be evolved to solve multiple tasks. It was noted that fitness increased as the length of evaluation time increased indicating that longer evolution time may lead to higher performing populations.

Embodied Distributed Evolutionary Algorithm (EDEA)

The Embodied Distributed Evolutionary Algorithm (EDEA) was presented in [32]. It is similar in theory to the approach taken in [22, 60, 59] but the actual implementation is quite different.

EDEA distributes its population across a group of robots so each robot contains a single genotype. Adaptation occurs through robot-to-robot interactions. In the presented work, each robot in the population was part of a connected network and could contact any robot within the population at any time. When potential mating partners are contacted a single individual must be selected to reproduce with. Robots in the mating process take on either a select role when initiating reproduction, or an eager role when accepting a reproduction request. A mating partner is selected using a binary tournament selection, and a weighted probabilistic decision (using the fitness of each individual and a selection coefficient) is made as to whether or not the process should continue. If reproduction continues, crossover and mutation are applied to create a new individual which replaces the genotype of the initiating robot.

When a new individual is activated, the robot begins carrying out the actions defined by the controlling genome. Fitness is calculated for the robot, but it cannot reproduce until a recovery time period has elapsed. This allows the new controller to adapt to its environment and to limit the effect the previous controller has on the fitness of the current one. After the recovery period is over, the robot can interact and reproduce with other robots in the population.

EDEA was tested in simulation on a number of test problems including phototaxis, fast forward, and collective patrolling, and the results of testing were compared to the performance of the encapsulated $\mu + 1$ ON-LINE algorithm on the same problems. The two algorithms were both found to be capable of solving the given tasks, and neither significantly outperformed the other, although EDEA was found to be less sensitive to changes in control parameters.

EVAG

In [29], an on-board, distributed algorithm based on the Evolvable Agent model (EvAg) [38] was presented. EvAg works to distribute computation over a number of processors as in the case of distributed on-board evolution.

Each robot in the population maintains a (1+1) population structure consisting of the best individual found so far, the "champion", and the individual currently being evaluated. Genetic information can be shared between robots within a local area using a newscast gossiping protocol to maintain network connections between robots. Along with the standard (1+1) population, each robot also maintains a cache of recently received genomes that are used for for parent selection during reproduction. Periodically, the robots take one genome from its local cache and contacts the robot from which the genome was created. These robots then exchange their cache information. Through this process genetic information is shared between all the robots in the population and leads to high performing solutions being present in the caches of multiple robots within the population.

The presented implementation used evolutionary operators similar to the $\mu + 1$ ON-LINE algorithm allowing performance between the two methods to be compared more accurately. During parent selection, parents are chosen using a binary tournament selection from the union of the local population and the cache of received genomes. Because the caches of robots are shared between robots, this selection is equivalent to selecting a random genome from the total population of solutions in all robots.

The EvAg approach was tested on an obstacle avoidance task, and its performance was compared to that of $\mu + 1$ ON-LINE, and to a panmictic version of EvAg where the robot network is fully connected. Comparing the panmictic and newscast variants of EvAg was done to determine the affect on performance of maintaining a peer-to-peer network between all robots. This is an important factor to the scalability of the newscast EvAg approach because in the real world fully connected networks are difficult or impossible to maintain. The performance of each approach was tested in simulation. Results showed that for a small population size of 4, the encapsulated and distributed approaches were not significantly different, but as the number of robots increased from 4 to 400 EvAg began to significantly outperform $\mu + 1$ ON-LINE. Stability of found solutions also increased as the number of robots increased.

The difference in performance between the newscast vs panmictic variants of the distributed algorithm became significant when the number of robots was set at 400 with the panmictic algorithm performing better. The cause of this difference was not determined, although it was hypothesized that the additional tuning parameters present in the newscast variant could contribute to the difference.

Overall it was determined that the distributed EvAg approach was effective in evolving controllers to solve the test task. Performance and stability of solutions increased as the size of the population increased indicating the the parallel nature of distributed evolution is able to effectively take advantage of increased population sizes.

odNEAT

The work in [45] presented the On-line Distributed Neuroevolution of Augmenting Topologies (odNEAT) approach for distributed, embodied robotic evolution. Rather than simply evolving weights for a controlling neural network, as is the common practice in evolutionary robotics, odNEAT is based on the NEAT method of neuroevolution which evolves both weights and the topology of the network itself. odNEAT is similar to the Real Time Neuroevolution of Augmenting Topologies (rtNEAT) [50] approach described in Section 2.4.1.

The population of odNEAT is distributed across a group of robots with each robot maintaining a list of genomes consisting of the robot's previous genotype, its current genotype, and all genotypes received from other robots in the population. The population is grouped using the niching scheme of NEAT, but rather than a global innovation number, a timestamp is used as the distinguishing feature to enable a decentralised genotype chronology. This niching scheme groups individuals into different species based on their topologies. If the topologies are the same, individuals belong to the same species. This allows evolution to occur while changing the structure of the evolved networks. Each individual also maintains a list of recent poor solutions, referred to as the taboo list, so that received genomes aren't brought back into the population once they are thrown out.

As each robot operates, it maintains both a virtual energy level and a fitness value. The energy level is a measure of the current robot controller's performance. Because performance evaluations in the real world are inherently noisy, the fitness of a genotype is considered as the average of the energy level sampled at regular intervals. When the energy level reaches zero, an intra-agent reproduction occurs. First, a random species is chosen from the robot's local population. Second, two parents are chosen from the selected species using a binary tournament selection. Third, crossover and mutation are applied to generate a new genome. This genome is then activated and allowed to operate during a recovery (or maturation) period allowing the robot to test its fitness and spread its genetic information before possibly being removed from the population. During operation each robot probabilistically broadcasts its active genotype to its local neighbors with a probability of being accepted defined as:

$P(event) = \frac{\bar{F}_k}{\bar{F}_{total}}$

with $\bar{F}_k$ defined as the average fitness of the local species k from which the current genotype belongs, and $\bar{F}_{total}$ defined as the average fitness of all local species. When this type of inter-species reproduction event occurs, the received genotype is first checked against the list of poor solutions to make sure it hasn't already been discarded. The genotype will also not be added to the local solutions list if it already exists in the local list, or if it lowers the combined fitness of the local population. Through this migration of genetic information, each population within a robot may closely resemble the total population of all robots, but it wont contain all information.

odNEAT was tested in simulation on a group aggregation task. Experiments were run with different population sizes to test the scalability of the approach. With a population of 5 robots, odNEAT was able to successfully evolve controllers to solve the test task, and performance increased with populations of 10 and 15. Performance remained steady with populations of 20, 25, and 30, but solution stability increased as the population size continued to increase. Testing was done to determine which parts of the algorithm contributed most to the overall performance, and it was found that the local population was the most important component, followed by the taboo list. All parts were found to be significant to performance.

Evolution of collective behavior using micro-robots

A distributed evolutionary algorithm operating in real-world micro robots was presented in [34]. This work focused on the stability in evolving collective behaviors in on-line, on-board evolutionary approaches. Robot behavior is controlled by a finite automata, and evolved using genetic programming. This approach was used to evolve obstacle avoidance behaviors so the stability of generated solutions could be tested. The experiments were carried out with Jasmine IIIp micro robots with a bi-directional communication range of 2-10cm. This allowed the robots to pass genetic information between one another when they were in close proximity.

The controlling automata was represented using a string genotype with evolutionary operators being applied to the generated phenotype string. A set of 10 predefined mutations were available and were grouped into 5 categories: toggle inactive transitions, remove a state, add a state, change a condition, change a state. Mutations were designed to be complete and to only cause small changes at a time. When needed, a random mutation was chosen. During crossover, the two selected parents simply copied the genotype with better fitness to generate an offspring. A more complex crossover operator was not used due to limited RAM memory on the micro robots. A fitness function was designed to keep the robots moving, but to avoid obstacles. Every 30 seconds the current fitness was halved to limit the affect of previous behaviors on the current fitness.

Six, 25 minute experiments were performed using 20 micro robots for each experiment. Every 15 seconds mutation was performed, and when robots became close enough to communicate with one another crossover was applied. These experiments generated several types of behaviors. A large number of robots were clustered and stuck in the corners of the testing arena. Some robots generated wall following behaviors in an attempt to avoid all collisions. Of the robots that did not become clustered in corners, about 33% showed a behavior similar to collision avoidance. It was suggested that limitations in the presented approach including poor cluster communication abilities, robots getting caught on connectors, and anomalies in sensor data may have decreased the overall fitness of the approach. It was concluded that developed behaviors could be preserved, but was noted that it is difficult to perform enough real world experiments to be significantly significant.

2.4.3 Hybrid approaches

Situated and Embodied Evolution

In [57] a hybrid approach was presented combining elements of both encapsulated and distributed evolution. The proposed method can be described as an island model in which evolution occurs within an individual robot and genetic information is passed between robots.

Within each robot a genetic algorithm is run on a sub population of individuals specific to that robot. This group of genotypes is referred to as the "gene pool." During each generation, a new individual is created by selecting two parents from the gene pool through roulette wheel selection, and then applying single point crossover and mutation. This new individual is then added to the “gene queue” where it waits to be activated and evaluated. Genotypes received from other robots are also placed into the gene queue so they may be reevaluated and spread through the population. Individuals are dequeued from the the gene queue, made active within the robot, and then assigned fitness. That individual is then placed into the gene pool if the maximum pool size has not been reached. If the maximum gene pool size has been reached, an individual is only added if its fitness is better than that of the least fit individual currently stored in the gene pool.

While evolution is taking place, genetic information is asynchronously transferred to other robots. The genotype to be transferred is selected from the gene pool using roulette wheel selection. At every time step there is a chance to send the genotype to another robot based on the fitness of the selected individual.

This approach was tested using six Khepera mini robots each controlled by a simple two layer neural network. The task they attempted so solve was one of collision avoidance and the fitness function rewarded individuals for not hitting walls or other robots while continuing to move. Four separate experiments were run. Three were done with robot sub population sizes of 1, 5, and 10 with genetic information being transferred. The fourth experiment used a robot sub population of size 5 but no genetic information was transferred. This was used as a benchmark with which to compare the effect of gene transfer between robots.

The best performing approach was the sub population of size 5 with genetic information being passed between robots. In general, all the trials that used gene transfer outperformed the trial without it. It was also noted that the optimal sub population size could be dependent on the number of robots and the particular task.

Kubic

A hybrid model was presented in [36] and was tested with two Khepera robots on the task of collision avoidance of both static and dynamic obstacles. Each robot consisted of N/2 individual chromosomes which encoded a simple controlling neural network.

The two robots each had a specific role of either the sender or the receiver. The sender would periodically broadcast 1/4 of its best individuals to the receiver. The receiver would then incorporate these individuals into its population during reproduction.

A new generation was created by taking 1/2 of the chromosomes from an existing robot and then generating the other 1/2 through crossover and mutation. In the case of the sender robot the highest performing 1/2 of chromosomes are passed to the next generation. The receiver bot passes 1/4 of its best individuals and the 1/4 of individuals received from the sender robot. In this manner, high performing individuals are preserved within the population of solutions. The fitness function worked to maximize a robot's average speed while minimizing the time spent turning and the time spent near other objects. This generates collision avoidance behaviors.

Experiments were run with population sizes of 12, 20, and 40 individual chromosomes. Each experiment consisted of five runs that evaluated highest and average fitness for each robot role. An individual run consisted of 20 generations of evolution.

It was concluded that using a hybrid evolutionary approach allows evolution of an acceptable controller to happen quickly using small robot group (2) and population sizes (10-20). Evolution times varied from two to many hours dependent on population size. It was noted that several limitations may have negatively influenced performance. First, it was possible for less fit individuals to be sent from the sender robot to the receiver thereby possibly lowering the fitness of the receiver. Secondly, communication between the robots wasn't very reliable with data loss rates of 5-15%. Thirdly, there is a lag time between sending, receiving, and testing individuals that may limit the evolution process.

Hybrid EVAG

The work in [29] also presented a hybrid approach based on the Evolvable Agent model (EvAg). In the hybrid version, rather than a 1+1 population structure a $\mu + 1$ population is maintained on board each robot. If $\mu = 1$, then the hybrid approach operates as the basic distributed EVAG, but if $\mu > 1$, and the list of received genomes isn't used, then hybrid EVAG operates as the $(\mu + 1)$ ON-LINE approach. All other characteristics of the hybrid EVAG approach are the same as the distributed version and are fully described in Section 2.4.2.

The performance was tested in simulation using e-pucks and compared against the performance of $(\mu + 1)$ ON-LINE and distributed EVAG. Performance was found to be better than $(\mu + 1)$ ON-LINE. The hybrid approach was consistently the highest performing approach, but it did not significantly outperform the distributed EVAG for any robot population size. Similarly to the distributed EVAG approach, stability for the hybrid EVAG increased as the number of robots increased.

2.4.4 Summary

An important aspect of evolutionary robotics focuses on evolving controllers for robotic agents. On-line evolution takes place during the operation time of the robot. On-board evolution takes place within the physical hardware of the robot. When evolution occurs on-line and on-board it allows the robot to adapt in real time to changes in the environment or task. On-line, on-board evolution can be done within a single robot, or can be carried out over a population of robots using a distributed evolutionary model. Considerable research on on-line, on-board evolution has been done, but typically has been carried out in simulation, or on real robots with limited processing power, leading to limited evolutionary models. Thus, there is a need for robotic hardware with greater computational power that can run full scale evolutionary models on-line, and on-board, rather than in simulation.

Chapter 3: A Training Set Protocol for On-Line, On-Board Evolutionary Learning from Demonstration

3.1 Abstract

In Learning from Demonstration (LfD) a trainer demonstrates desired behaviors to a robotic agent. LfD is useful for allowing non-programmers to train robots. However, it has rarely been used with evolutionary learning techniques because of the computational requirements of running an evolutionary process on-board a robot and on-line, i.e. in parallel with the demonstration process. We present an on-line, on-board evolutionary LfD system capable of evolving robotic neurocontrollers on a low cost robotic platform. This system was proven capable of evolving controllers capable of following a colored path.

A feature of on-line LfD that may have significant implications for the implementation of evolutionary approaches is that the training set of desired behaviors is built incrementally over the course of multiple behavior demonstrations. If the initial training set lacks important cases, evolution may discard solutions that will eventually be useful and become stuck on local optima. An untested question for on-line, on-board evolutionary LfD is how large should training sets be before evolution begins? Thus, in addition to demonstrating a practical implementation of on-line, on-board evolutionary LfD, our results suggest that it can be beneficial to begin evolution with a small initial training set.

3.2 Introduction

For humans to regularly, easily, and reliably work cooperatively beside robots, it is necessary that robots quickly adapt to changes in problem, environment, or goals. Without this, humans quickly become frustrated with robots whose behaviors are overly simple, brittle, or unable to adapt to change. This frustration will severely limit humans willingness to interact closely with robotic systems.

Learning from Demonstration (LfD) [a] is a common and proven approach for allowing non-programmers to easily train robots. In LfD a human trainer demonstrates desired behavior and the agent records that behavior to build a training set that it learns from. LfD allows a human trainer to quickly build a training set for a particular task simply by repeatedly demonstrating the task and does not require any programming skills or specialized robotics knowledge. Because of this, LfD shows great potential as a means of training robots for a variety of unique tasks without specialized knowledge of robotic hardware or software.

Evolutionary robotics refers to the evolution of robotic controllers. Robots that evolve during runtime and within their own hardware are said to be evolving on-line and on-board. Evolutionary approaches can allow robots to adapt to tasks or environments during runtime operation, and are particularly desirable in complex and dynamic environments where manually designing controllers ahead of time is not a viable option.

While both LfD and evolutionary approaches have been effective with previous robotic systems, the two approaches have rarely been used together. The first goal of this paper is to demonstrate an on-line, on-board evolutionary LfD system that can be run on an affordable robotic platform. As a test problem, the robot evolved an on-board population of neurocontrollers to follow a distinctly colored path. The evolution is executed on-line while the trainer drives the robot on the path, giving the robot training cases. Our results confirm that the robot can conduct training and evolution simultaneously, and that a few minutes of training is sufficient to both build a reasonably sized training set and evolve reasonably successful solutions.

However, a critical feature of LfD is that the training set is built incrementally -

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

each new behavior demonstration adds to the training set. Given the time required for a human to demonstrate a task, there is often time to run several generations of an evolutionary algorithm in the background between observations of demonstrated behaviors. For example, in the current application an average of three generations of the GA can be completed between demonstrations. Thus, for maximum efficiency learning needs to be done, or at least initiated, on-line, i.e. during the demonstration period. Other research has shown that incremental fitness functions can be beneficial if they are designed to guide the evolutionary process. However, in LfD applications, the training set is likely to be built in an ad hoc function.

Thus, a critical question to the implementation of on-line, on-board evolutionary LfD algorithms is how the ad hoc, incremental construction of a training set influences the evolutionary process. In particular, there is a risk that an initial training set containing only a few demonstrated behaviors will drive evolution to a local optima. The second goal of this research is to examine whether delaying the evolutionary learning process until some minimum training set size has been reached is beneficial. Our results suggest that delaying the evolutionary process until a number of training examples have been generated is important to successful evolution, and that partially overlapping the collection of training data and evolution does reduce the total time required to evolve a solution, while still leading to successful solutions.

3.3 Related Work

In this section we review recent work in evolving robot neurocontrollers that motivates our push to on-board on-line evolution. We then discuss the previous work in Learning from Demonstration as a method for training robots by demonstrating desired behaviors.

3.3.1 Embodied Evolution of Robot Controllers

Research into evolutionary robotics has focused on optimizing robotic controllers during off-line evolution, in which the robotic agent is evolved in simulation, and the best solution is transferred to a physical robot when evolution is completed [32]. This approach works well for some problems, but has several important drawbacks. Because evolution takes place in simulation, evolved controllers are adapted to a simulated environment rather than the real world and therefore often suffer from decreases in performance [14, 35]. Furthermore, exploring in isolated hostile environments such a off-world or in mines or caves might best be done by distributed on-board systems where distribution of learning assets is critical to team survival. To create controllers that perform well in dynamic real-world environments, evolution must be embodied within the robotic hardware and take place during real-time operation.

Watson et al performed some of the earliest research using a distributed, on-board evolutionary algorithm to evolve neural networks [59, 22] to perform a phototaxis task. A typical experiment used eight robots, each controlled by a microcontroller. 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.

Considerable research has been done using s-bots as part of the swarmbots project in Europe [17]. This research has included the evolution of neural networks for phototaxis and hole-avoidance [15]. Most of the evolution was actually done in simulation, but the evolved networks were tested successfully in physical robots.

More recently Eiben and Haasdijk et al. have performed a number of experiments with on-board, distributed evolution, although mostly using the Player/Stage system [b]

[b]Player/Stage system http://www.playerstage.sourceforge.net/

as a simulation environment [27, 29, 32, 28] 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 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 [29].

Capi and Toda demonstrated true on-board, on-line evolution of neural controllers for robot navigation using a custom built robot [14]. Similarly to the approach proposed in this work, their system partitioned collected images into regions that were used as inputs to the evolving neural controller (see Section 3.4.3 and Section 5.1). Using this system their robot learned to successfully navigate in indoor environments.

Although there has been considerable interest in on-line, on-board evolution, most of the research has actually been done in simulation. Thus, an important aspect of this research is demonstrating true on-line, on-board evolution using physical and affordable robots.

3.3.2 Learning From Demonstration

In general, Learning from Demonstration is any approach to machine learning in which a robot or other computerized agent learns a task by observing behaviors demonstrated by a teacher, most often a human [5, 44]. LfD has been applied to a wide range of tasks with significant success, including learning gestures and movement sequences [3, 7, 12], facial expressions [9], and communication [4]. Most important to this research, it has been used with significant success for mobile robot navigation [13, 39, 47, 46, 54]).

There are also a few cases where LfD has been successfully combined with Evolutionary Computation (EC). 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 [56]. Aler, Garcia, and Valls used evolutionary techniques and demonstrations of correct behavior to incrementally correct a team of simulated robosoccer players [1]. Similarly, Sullivan and Luke used a LfD system to train agents within the RoboCup competition [55]. We have demonstrated that our robots can learn both object following [49] and path following [16] tasks from demonstration.

In addition, research shows that the degree of correspondence between the teacher and the learner can significantly influence the speed and effectiveness of LfD, 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 [5]. Thus, in this research a remote control (a second smartphone) is used to control the robot during learning from demonstration (see Section 4.3.4).

3.4 Overview of on-line, on-board, evolutionary Learning from Demonstration system

The system presented here combines Learning from Demonstration (LfD) with a genetic algorithm for the training of mobile robots. The robots observe demonstrated behaviors to build a training set of inputs and their corresponding desired actions. A genetic algorithm then evolves a population of potential neural network controllers that work to replicate the demonstrated behaviors. When evolution is finished, the evolved controller that best replicates the demonstrated behaviors is made activate and autonomously controllers the robot.

3.4.1 Robot

The robot used in these experiments was designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles [48]. The robot consists of three basic components: an Android smart phone, an Arduino microcontroller, and

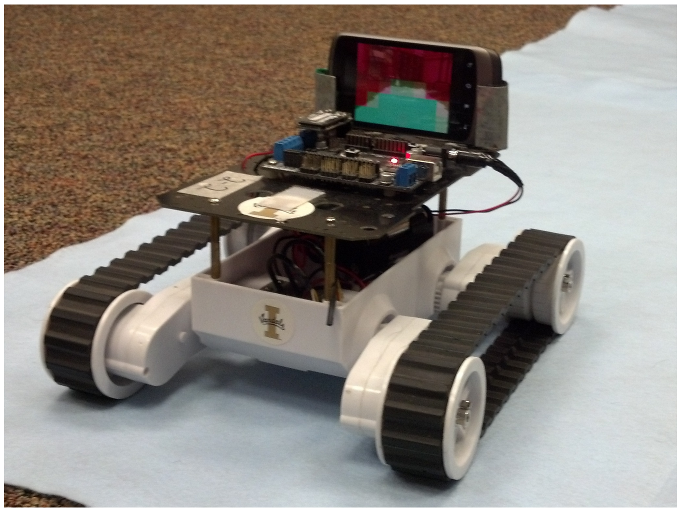


Figure 3.1: Image of the used robot platorm. The smartphone can both receive wireless commands from a remote phone via Bluetooth, and send commands to an Arduino type microcontroller also using Bluetooth. The microcontroller interprets received commands, and can forward movement commands to the robot's motors. Sensing, processing, and on-board evolution/learning is done within the robot smartphone.

a robot platform. The smart phone used in these experiments was the HTC Nexus One, which has a 1GHz Qualcomm Snapdragon processor, with 512 MB of RAM, and a Qualcomm Adreno 200 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[c], which is a small (24.5cm by 22.5cm by 7.4cm), tracked vehicle controlled by two DC motors and is capable of running continuously for several hours while carrying the microcontroller and smart phone. The complete robot is shown in Figure 3.1, and a diagram of the robot's communication flow is showing in Figure 3.2. A second smart phone runs a remote control app that is used to allow a user to control the robot for learning from demonstration via a second Bluetooth connection.

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

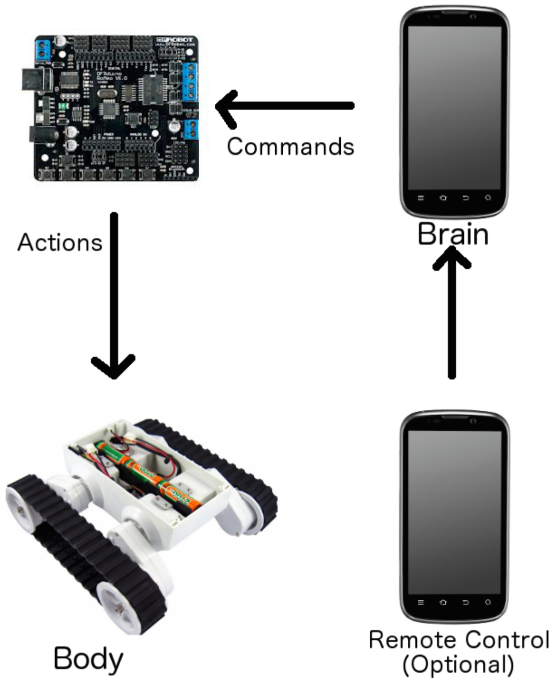


Figure 3.2: This image depicts each hardware component of the robot, and how it communicates with other components in the system.

3.4.2 Road Library

The first step in this system is to build a library of safe road examples stored as a list of color histograms. This library is constructed at the beginning of training by giving the robot ten drive commands on the training track. It is assumed that during this library building step, everything directly in front of the robot is safe road, so histograms are constructed from a small sub image taken from the bottom center of the camera image. Each histogram is saved to the road library so it can be used during training and autonomous operation. The road library in these experiments was limited to a size of ten.

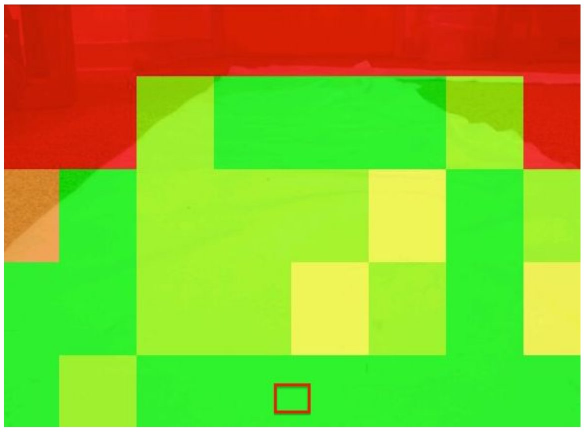


Figure 3.3: Processed image colored to indicated output value. Green areas represent high road probability values, and red represents very low probabilities. The area outside of the path is nearly all red, while the path area generally has high values.

3.4.3 Image Processing

To generate inputs for the controlling neural network, images from the smartphone's camera are taken and processed. Processing an image produces 40 real values between 0.0 and 1.0 that represent an image sub region's probability of being a known, safe road type.

Image processing is done in several steps. To start, an image is taken from the camera and split into 40 uniform sub regions. A color histogram is then calculated for each sub region. Using this histogram, a back projection calculation is done for each histogram in the road library.

The back projection calculation determines the probability that the color distribution of a pixel from an input histogram matches the color distribution of some comparison histogram. In this case, the input histogram belongs to the sub region, and the comparison histograms are taken from the road library. The resulting probability is then assigned to the corresponding pixel in an output image.

The resulting image from each of these back projection calculations is averaged, and the highest average value is assigned as the probability for a sub region of belonging to a safe, known road type from the road library. Figure 3.3 shows a processed image colored to represent the processed output values. Once all 40 sub regions have been assigned a value, those 40 values are used as inputs for the controlling neural network.

3.4.4 Neural Network Controller

This section describes the architecture of the neural network used to control the robot. Note that for these experiments, because the focus of the research is on the evolutionary LfD system and not neural network design, the NN architecture is fixed. However, there is no reason that the network architecture couldn't be evolved as well, using NEAT [52], HyperNEAT [51], or a similar algorithm, as long as the number of input and output nodes remained fixed.

In this work, input nodes are given the values resulting from image processing, and the robot's actions are determined by the neural network's output node with the highest activation level. This is a common approach to evolving NN controllers for autonomous robots [14]. In cases of a tie on the output nodes, no command is sent to the robot. However, as the outputs of the nodes are real values, this is an extremely unlikely event, which never occurred during the experiments.

The neural network has forty input nodes, one for each sub region in a processed image. It has a single hidden layer consisting of five nodes, and an output layer with three nodes, which correspond to the actions: *left*, *right*, and *forward*. Each node uses a sigmoid activation function. In total, the network has 223 weights between nodes(40 input nodes to 5 hidden nodes = 200, 5 hidden nodes to 3 output nodes = 15, and 8 bias nodes). The 223 weights of a controlling network are evolved during the training process as described in Sections 3.4.5 and 4.3.4.

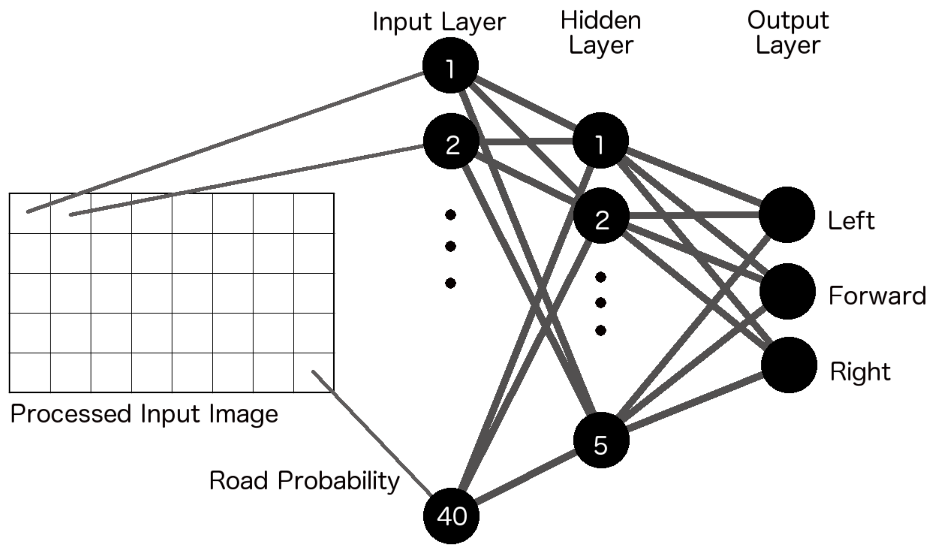


Figure 3.4: This image shows the architecture of the neural network used to control the robot. Inputs are taken from a processed image, fed through the neural network, and the output node with the highest activation value determines the action the robot will take.

3.4.5 Evolutionary Algorithm

An important goal of this project is to test the feasibility of using smart phones for on-line, on-board, evolutionary learning from demonstration. For these initial investigations, we intentionally chose to use a simple, but general, evolutionary model. We used a standard generational genetic algorithm (GA) to evolve neural network weights. Individuals are represented as an array of 223 real valued weights; one for each weight in the neural network. Initially the weights were randomly set in the range -1.0 to 1.0. Each individual's fitness was initially assigned to 0.0, and is only calculated with the construction of the next generation, meaning that the first generation uses a random selection.

Two individuals are selected as parents using tournament selection of size three. A single offspring is produced and undergoes crossover and mutation before being added to the next generation's population. Uniform crossover was used, and mutation was applied at a rate of 0.2 with the mutation amount uniformly distributed between -0.1 to 0.1. The

Population size	40
Generations	100
Mutation rate	.2
Crossover rate	100% Uniform, 50% chance to exchange weights
Selection method	Tournament(size 3)
Rounds of Training	5 for each experimental condition

Table 3.1: Summary of the evolutionary algorithm parameters.

parameters of the evolutionary algorithm are summarized in Table 3.1.

Fitness was determined by comparing the trained action given for a set of road probabilities to the output action generated by each individual network given the same road probabilities. The training cases are divided into category based expected actions (Left, Right, Forward). 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 one turn command given. The algorithm for the fitness function can be seen in 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 $\geq$ 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
```

Assigned fitness is scaled based on the difference between the highest activated output value and the sum of the remaining output values. A threshold of 0.4 was used to determine whether full fitness was assigned for getting a test case correct. This value was chosen arbitrarily because it was in the middle of the possible range of threshold values, and its effect was not tested. When the correct action is chosen, if the difference between the highest output value and the sum of the remaining values is greater than 0.4 then full fitness will be assigned to that training case. Otherwise, positive fitness is still assigned, but it will be scaled down based on how much below the 0.4 threshold the difference is. This process rewards network that have some sense of "confidence" about their chosen actions.

3.4.6 Training Process

During human demonstration, a human drives the robot, following a path defined by colored felt on a different colored background. The human uses a remote control with discrete forward, left, and right actions. While the human is driving, a training set of demonstrated behavior is constructed. Each training case in the training set consists of 40 processed image probabilities, and the corresponding demonstrated behavior (forward, left, or right).

While the human operator is demonstrating behavior, the evolutionary algorithm is evolving individuals to best replicate the behavior contained within the current training set. Training continues until 100 generations of the genetic algorithm have been completed, at which point the most fit individual is made active as the robot's controller.

3.4.7 Autonomous Operation

During autonomous operation the robot uses the best evolved neural network for navigation control. Input images are continuously compared against the road library to

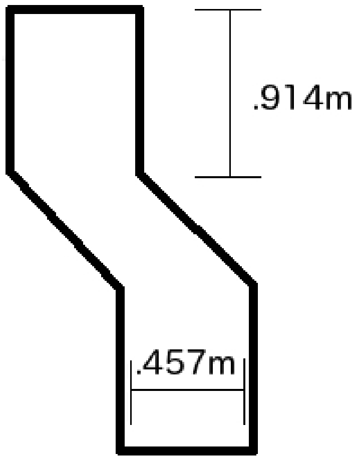


Figure 3.5: The training track for these experiments. The track consists of one 45° left corner, one 45° right corner, and three straight sections. Training is always started from the same end of the track, and the robot is always trained to drive to the opposite end of the track. It is then turned around and driven back along the track. This process is continued until evolution is complete.

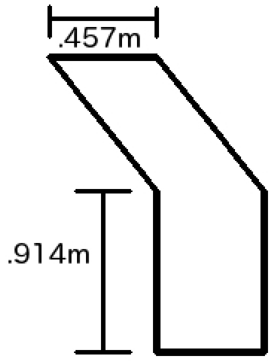
generate road probability values as inputs for the neural network which then returns its trained action.

3.5 Experimental Design

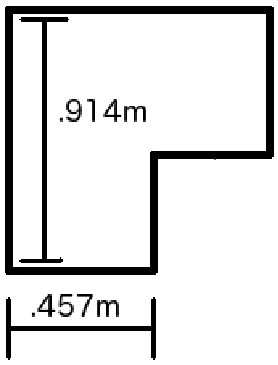
This section describes the experiments performed to determine the effectiveness of the proposed on-line, on-board, evolutionary Learning from Demonstration system.

3.5.1 What was tested

To measure the performance of the evolved neural network, the robot's ability to navigate a path with different corner types is measured. This measure was also used to test the generality of the approach as the robot is trained on only a two corner types(left 45°, right 45°), but tested with four corner types. The four test corners were used: left 45°, right 45°, left 90°, and right 90°. Figures 3.5 and 3.6 show the training and test tracks respectively. Each trained NN was tested on each corner three times, so the highest



(a) Testing track 1.



(b) Testing track 2.

Figure 3.6: Figure 3.6a shows 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 3.6b shows 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.

performance possible for each NN is 12.

3.5.2 Testing procedure

Each experiment was conducted as follows:

1. Build road library
2. Construct initial training set (see Section 3.5.3)
3. Demonstrate how to drive on the training track until 100 generations have been evolved
4. Place the robot in autonomous mode
5. Allow the robot to autonomously navigate both of the testing tracks three times in each direction, recording how many times it successfully navigated each corner.

3.5.3 Experimental Conditions

To test the effect on performance of the size of the initial training set, four initial starting conditions were tested:

1. 1 Training Case - Evolution begins as soon as the first training case (the first demonstration) is received. Evolution and demonstrations continue, in parallel, until all 100 generations are complete.

2. 1 of Each Training Case - Evolution begins as soon as one training case for each action (left, right, and forward) is collected. In this condition, one example of each behavior was demonstrated to the robot immediately and then it was driven on the training track normally to generate additional training cases during evolution.

3. 3 of Each Training Case - Evolution begins as soon as three training cases of each action (left, right, and forward) are collected. In this condition, three examples of each behavior were demonstrated to the robot immediately and then it was driven on the training track normally to generate additional training cases during evolution.

4. Offline - Evolution begins *after* all of the training cases have been collected. Thirty five training cases were used, as this is the average number of training cases that were collected in the '1 of each' case.

Note that in the first three test conditions evolution is happening in parallel with the demonstrations. So, the set of training cases (demonstrated input-behavior pairs) increases throughout the evolutionary process. The goal is to determine whether having a small initial training set hinders the evolutionary process.

	Training	45° Left	45° Right	90° Left	90° Right	Total Out of 12
1 Training	1	3	0	2	0	5
	2	2	2	2	0	6
	3	2	2	3	1	8
	4	2	3	3	1	9
	5	3	2	3	0	8
	Total	$\frac{12}{15}$ (80%)	$\frac{9}{15}$ (60%)	$\frac{13}{15}$ (87%)	$\frac{2}{15}$ (13%)	$\frac{36}{60}$ (60%)
1 of Each	1	0	3	1	3	7
	2	3	3	2	3	11
	3	2	3	0	1	6
	4	2	3	1	2	8
	5	2	2	3	2	9
	Total	9/15 (60%)	14/15 (93%)	7/15 (47%)	11/15 (73%)	41/60 (68%)
3 of Each	1	3	2	3	0	8
	2	3	1	3	1	8
	3	2	2	2	3	9
	4	3	2	3	0	8
	5	2	3	1	2	8
	Total	13/15 (87%)	10 (67%)	12 (80%)	6 (40%)	41/60 (68%)
Offline	1	2	3	3	3	11
	2	1	3	1	3	8
	3	3	2	3	1	9
	4	0	3	2	3	8
	5	2	3	2	0	7
	Total	8 (53%)	14/15 (93%)	11/15 (73%)	10 (67%)	43/60 (72%)

Table 3.2: Results of testing evolved controllers on 45 and 90 degree corners. Three tests were conducted for each corner, per test. The data shows that as the size of the initial training set increases, so does the total number of corners successfully navigated.

3.6 Results

Table 3.2 shows the number of successes (for each of the four test corners) for each evolved ANN. First, and perhaps most importantly, the results show that the ANNs learned to successfully navigate corners, including sharper corners than were in the training data, at least 60% of the time. This demonstrates that evolutionary learning from demon-

	Training	Avg	Best
1 Training Case	1	0.66	0.64
	2	0.43	0.39
	3	0.25	0.18
	4	0.41	0.36
	5	0.27	0.2
	Average	0.4	0.35
1 of Each	1	0.06	0.05
	2	0.04	0.03
	3	0.02	0.01
	4	0.16	0.13
	5	0.19	0.13
	Average	0.09	0.07
3 of Each	1	0.18	0.18
	2	0.21	0.19
	3	0.13	0.11
	4	0.07	0.05
	5	0.12	0.1
	Average	0.14	0.13
Offline	1	0.02	0.04
	2	0.01	0.01
	3	0.07	0.09
	4	0.0	0.0
	5	0.0	0.0
	Average	.03	.02

Table 3.3: The average and best fitness of each round of training. The data shows an improvement in fitness (through minimization) in both the average and best cases as the number of training cases gathered before evolution begins increased.

stration in a real world application is feasible. The training process is simple, requiring no knowledge of software and taking less than 5 minutes to train a robot for this simple problem. This method of learning was capable of producing robotic controllers capable of navigating the same type of corners it was trained on (45°) 77% of the time after only 100 generations. The robot was able to navigate 90° corners, which it wasn't trained for, 60% of the time in the '1 of each' and '3 of each' training sets, showing some generalization of evolved behaviors. It is very likely that with more sophisticated evolutionary techniques, more generations, or more training cases, much better results could be achieved.

Second, our results show an increase in performance when evolution is started with more than a single training case. Both the '1 of each' and '3 of each' cases performed better than starting evolution on the first demonstration. Notably, the '1 training case' condition performed extremely poorly on the right 90° corner (Table 3.2). For our training track the robot often did not reach the first right hand corner until significantly into the training process, e.g. after 50 or more of the 100 generations had been completed. Thus, for over half of the evolutionary process there was no training case that required a right turn. This is a likely explanation for why it performs so poorly on the right 90° corner. However, the results also don't show a difference in performance between the '1 of each' and '3 of each' conditions suggesting that simply having a single example of each type of action when evolution starts is sufficient to improve performance of the evolutionary algorithm. Finally, the off-line evolution data shows the best overall performance. This isn't surprising as it suggests that having all of your training examples at the start of evolution is the best way to evolve a controller that can replicate the demonstrated behaviors.

Table 3.3 shows the best and average fitness for each of the trials and conditions. With the exception of the '1 training case data', the fitness tends to be better than the actual test performance. This suggests that these cases could be over-fitting the training data. Thus, a possible advantage of on-line training is that it reduces the risk of over-fitting by continually adding new training cases.

3.7 Conclusions

Our results show that on-line, on-board evolutionary Learning from Demonstration, implemented on low cost, but computationally powerful robots, is a practical approach to training robotic agents. Using this approach we were able to train robots to complete a simple path following task. The results show that evolutionary LfD performed on-line, i.e.

evolution occurs in parallel with the demonstration process, performs comparably, but slightly worse than, off-line evolutionary LfD in which evolution is delayed until after all of the demonstrations are complete. However, the results also suggest that this difference can be minimized by “seeding” the training set with a small number of demonstrations before initiating evolution, and that the on-line approach may reduce over-fitting by continually expanding the training set with new examples, which, for more complicated problems, is likely to lead to better performance in the on-line case. The major advantage of the on-line approach is that it reduces the total time for the learning process by parallelizing demonstrations and learning.

Chapter 4: Real World Applications of On-Line, On-Board Evolutionary Learning from Demonstration

4.1 Abstract

Learning from Demonstration (LfD) is a useful approach for teaching robotic agents new behaviors because it requires little specialized knowledge of robotic hardware or of computer programming. Evolutionary learning techniques have also commonly been applied to robotic learning tasks, but the two techniques have rarely been used together due to the computational requirements of running an evolutionary process on-line, and on-board a robot during the LfD process. This work presents an on-line, on-board, evolutionary learning from demonstration protocol that enables two different low-cost robotic platforms to learn to follow two distinct path types. In addition, the robots are capable of recording their GPS location during autonomous operation which has a wide variety of practical applications such as autonomous trail mapping, and marking objects of interest in a natural disaster response application.

4.2 Introduction

Systems allowing human/robot interactions have increased over the years but these aren't always simple to design or use. For truly effective human/robot interaction, robots must be easy to train and must be general enough to be applied to a wide variety of tasks without needing to be reprogrammed. Without these requirements being met, humans will likely choose not to use robotic agents deeming them too cumbersome to use or too limited in their abilities.

Evolutionary robotics is a field of research focused on applying evolutionary learning within robotic agents in order to generate robots that can successfully complete a variety of complex tasks. The ability of evolutionary approaches to find effective solutions makes this approach very general and can allow a single robotic controller to complete different tasks through constant adaptation. Evolutionary learning is particularly well suited to tasks and environments that are dynamic or are too complex to be simulated accurately. In these cases, the power of evolution to adapt to new environments and new tasks allows a single robotic agent to be used in a variety of situations without needed to be redesigned or reprogrammed, just retrained. This is a great benefit for tasks such as natural disaster response where an autonomous robot could largely benefit the relief effort, but where time does not permit custom development of a new robotic controller.

Learning from demonstration (LfD), also known as Learning from Imitation, is an approach to robotic learning in which a robotic agent learns new behaviors by observing actions demonstrated by a trainer. This trainer generally has special domain knowledge about the task being demonstrated, but doesn't need to know anything about the internal workings of the robotic controller or hardware. The trainer simply performs desired actions, and the robot records those actions, along with the corresponding inputs, to build a training set of desired behaviors. The robot then learns to replicate the behaviors present in the training set. The method the robot uses to learn behaviors can vary, but does not matter to the demonstrator. Because of the simplicity of the demonstration process, a human trainer can quickly build a training set of desired behaviors. The approach is also general enough that it has been applied to a variety of tasks such as robot navigation [13, 39, 47, 46, 54] learning facial expressions [9], and robot soccer [53, 25, 55].

By combining the ability of evolutionary approaches to find solutions to complex problems, and the simplicity of training agents with LfD, a powerful, yet simple, evolutionary LfD system can be created. This approach was taken be Sullivan and Luke to train robotic soccer players for the RoboCup competition [55]. Evolutionary LfD works to solve the problems in human/robot interactions by making a simple and flexible system that can be applied to a variety of tasks without specialized knowledge or robotic hardware or software.

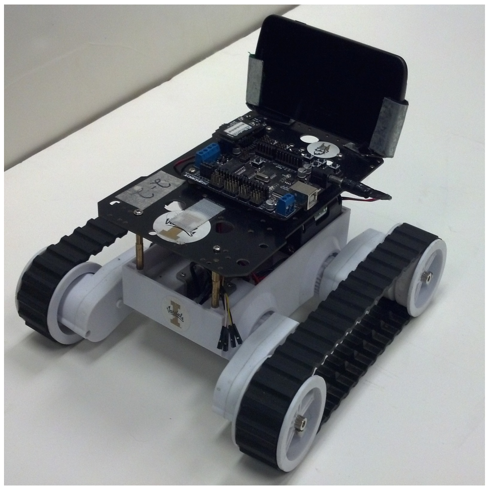
This chapter presents an on-line, on-board evolutionary LfD system capable of training a mobile robot to follow paths. Two different mobile robot platforms are used to test the training system's ability to teach the robot to follow two distinct paths: an urban environment brick path, and a tree bark path within a park. The percent of the total path distance traveled for each path was recorded. In addition to learning to autonomously follow a path, the robots are capable of storing their GPS position which allows them to build a map of their traversed path.

4.3 Methods

The following sections describe the robots, training system, and experimental procedure used in these experiments.

4.3.1 Robotic Platforms

Two robots were used in these experiments. Both robots were designed and built at the University of Idaho using Commodity Off The Shelf (COTS) design principles [48]. Each robot is comprised of three components: an Android smart phone, an Arduino-type microcontroller, and a robot platform. The phone performs all data processing and decision making, and then sends action commands to the Arduino board, which signals the robot platform to carry out the specified action. The robot platform also serves to carry the smart phone and microcontroller so all sensing and processing is done on the robot itself. The smartphone used in these experiments was the Samsung Galaxy Nexus which has a 1.2 GHz dual-core ARM Cortex-A9 processor, 1 GB of ram, a 307 MHz PowerVR SGX540 GPU, and a 5 MP camera. The two different platforms are described below in Sections 4.3.1 and 4.3.1.


(a) Rover 5 Platform

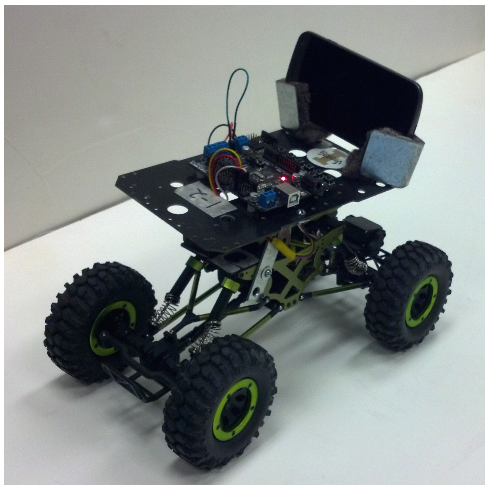

(b) Rock Crawler Platform

Figure 4.1: The robot platforms used for these experiments.

Rover

The first platform is the Rover 5 Tank chassis[a], which is a small (24.5cm by 22.5cm by 7.4cm), tracked vehicle controlled by two DC motors and is capable of running continuously for 3-5 hours while carrying the microcontroller and smart phone. This platform is shown in Figure 4.1a

Rock Crawler

The second platform is a remote control rock crawler platform[b]. It measures 31.4cm by 17.5cm by 14cm, is controlled by a dc motor for power/movement, and a servo motor for steering, and has a steering angle of 40-50 degrees. It is capable of operating for 3-5 hours while carrying the smartphone and microcontroller. This platform is shown in Figure 4.1b.

[a]Rover 5 Tank chassis http://www.dfrobot.com
[b]RC Rock Crawler Platform http://www.nitrorcx.com

4.3.2 Procedure Overview

These experiments were each broken into three distinct phases: initial road library construction, evolution during demonstration, and testing. An initial road library construction step told the robot which type of road to recognize. During the demonstration phase a human operator demonstrated behaviors to the robot which evolves a population of neurocontrollers to replicate the demonstrated behaviors. In the testing phase, the best neurocontroller was used to autonomously drive the robot, and its ability to follow the trained road type was tested. More detail is given for each phase below.

4.3.3 Road Library Construction

The first phase in each experiment was to build a library of safe road examples stored as a list of color histograms. This library was constructed by giving the robot ten drive commands on the current path making sure to give commands that take the robot through portions of the path with different surface or lighting conditions. It is assumed that during this library building phase, everything directly in front of the robot is safe road, so the color histograms are constructed from a small sub image taken from the bottom center of the camera image. Each histogram is saved to the road library so it can be used during training and autonomous operation. This process isn't considered to be a form of "machine learning;" it simply remembers the examples of safe road that it is shown.

4.3.4 Evolutionary Learning from Demonstration

The robot learns by observing demonstrations from a human operator, and then evolving a controller to best replicate those demonstrated behaviors.

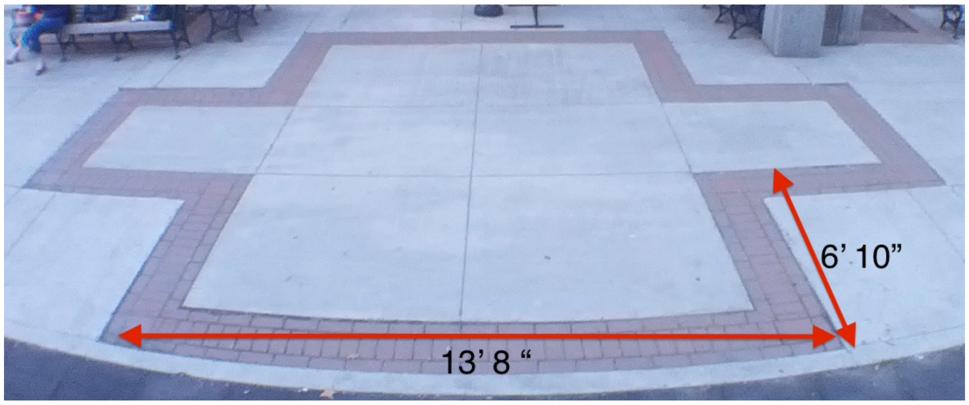


Figure 4.2: This is the urban environment path used in these experiments. The robot was trained on the brick path, and attempted to navigate around the full path without driving off onto the surrounding concrete. This path was challenging due to the reflective concrete surface, differences in lighting that occurred around the path, and the sharp corners.

Training Process

During human demonstration, a human drives the robot on path in an outdoor environment. For these experiments, two different path were used: a brick path surrounded by concrete, and a bark path surrounded by grass (see Figures 4.2 and 4.3). The human uses a remote control with discrete forward, left, and right actions. While the human is driving, a training set of demonstrated behaviors is constructed. A demonstrated behavior is stored in the training set as a training case. Each training case consists of 40 processed image values, obtained as output from image processing (described in Section 4.3.4), and the corresponding demonstrated behavior.

While the human operator is demonstrating the desired driving behavior, a genetic algorithm is evolving individuals to best replicate the behavior contained within the current training set. Training starts after 3 demonstrations of each behavior have been added to the initial training set, and continues until 200 generations of the algorithm have been completed, at which point the most fit individual is made the robot’s controller for testing.

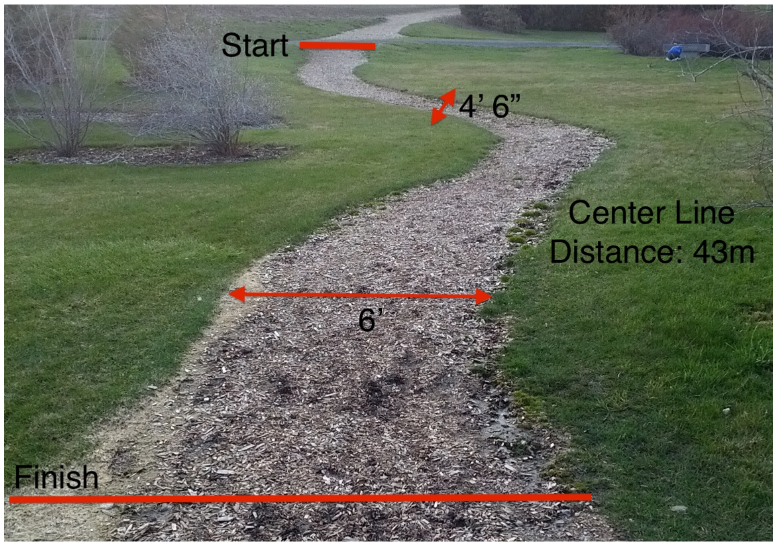


Figure 4.3: This is the park environment path used in these experiments. The robot was trained to navigate the bark path and to avoid driving off into the surrounding grass. This path was challenging due to a nonuniform path surface composed of lightly colored bark, and darkly colored mud. The path also changed widths and in some places has a less clearly defined edge.

Image Processing

To generate input values for the controlling neural network and training cases, images from the smartphone's camera are taken and processed. Processing an image divides the image into 40 sub regions and produces 40 real values between 0.0 and 1.0 that represent each sub region's probability of being a known, safe road type.

Image processing is done in several steps. First, an image is taken from the camera and split into a 5×8 grid of 40 uniform sub regions. Second, a color histogram is calculated for each sub region in the image. Third, using each created histogram, a back projection calculation is done for each histogram in the road library. The back projection calculation determines the probability that the color distribution of a pixel from an input histogram matches the color distribution of a comparison histogram taken during the road library construction phase. In this case, the input histogram belongs to the image sub region, and the comparison histograms are taken from the road library. The probabilities calculated during the back projection are assigned to a grey-scale image of the same size as the input histogram, where each pixel represents the probability for the corresponding pixel in the input image.

The output image from each back projection calculation is then split into 40 sub regions. Each sub region is assigned the average pixel value from that sub region. The highest average value is assigned as the probability of a sub region belonging to a safe, known road type from the road library. Once all 40 sub regions have been assigned a value, those 40 values are used as inputs to the controlling neural network or as part of a training case.

4.3.5 Neural Network Controller

The robot is controlled by a simple neural network with a fixed topology. The network has forty input nodes, one for each sub region in a processed image, 5 hidden layer nodes, an output layer with three nodes corresponding to the robot's available actions: *left*, *right*, and *forward*, and 8 bias nodes. Each node uses a sigmoid activation function. In total, the network has 223 weights between nodes (40 input nodes to 5 hidden nodes = 200, 5 hidden nodes to 3 output nodes = 15, and 8 bias nodes).

Input nodes are given the values produced during image processing, and each robot action is determined by the neural network's output node with the highest activation level. If more than one output node has the same activation level, no command is sent to the robot. However, because the output values of the nodes are real values, this is an unlikely event.

4.3.6 Genetic Algorithm

To evolve neural network weights, a standard generational genetic algorithm was used. Individuals are represented as an array of 223 real valued weights; one for each weight in the neural network. Weights were initially set in the range -1.0 to 1.0. An individual's

Population size	40
Generations	200
Mutation rate	.2
Crossover rate	100% Uniform, 50% chance to exchange weights
Selection method	Tournament(size 3)

Table 4.1: Summary of the genetic algorithm parameters.

initial fitness was set to 0, and is only recalculated during the construction of each new generation. This results in the first generation using a random selection.

Two individuals are selected as parents using tournament selection of size three. A single offspring individual is produced via crossover and mutation operations before being added to the next generation's population. Uniform crossover was used, and mutation was applied at a rate of 0.2 with the mutation size uniformly distributed between -0.1 to 0.1. The parameters of the evolutionary algorithm are summarized in Table 4.1.

Fitness was determined by comparing the human's demonstrated action given for a processed image to the output action generated by each individual network given the same processed image as input. The training cases are divided into categories based on their expected actions (left, right, forward), and the fitness reward for getting an instance correct (eg. a left action correct) was normalized by the number of actions in that category. An individual 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. Fitness was calculated this way because during training, there were generally ten forward commands to every one turn command given. Normalization ensures that an individual must do equally well at performing all demonstrated behavior types, and can't just focus on replicating the action that was most often demonstrated.

Assigned fitness was scaled based on the difference between the highest output value and the sum of the remaining output values. An arbitrarily chosen threshold of 0.4 was used to determine whether full fitness was assigned for getting a test case correct. When the correct action was chosen, if the difference between the highest output value and the sum of the remaining values is greater than 0.4 then full fitness will be assigned to that training case. Otherwise, positive fitness was still assigned, but it was scaled down based on how much below the 0.4 threshold the difference is. This process rewards networks that have some sense of "confidence" about their chosen actions.

4.3.7 Testing

To test each trained controller, the robot was placed on a path at the same starting location each time. It was then allowed to operate autonomously in an attempt to follow the path. The percent of the total path distance that the robot traveled was recorded. If the robot's body went totally off the track but was still following parallel to the path or immediately drove back on to the path it was allowed to continue. If the robot left the path, and started driving away from the path it was considered a failure. Testing was done in two different environments using a different robot platform for each one. The tracked rover platform was used when testing on an urban environment path (see Figure 4.2), and the rock crawler platform was used when testing on a path in a park environment (see Figure 4.3).

While the robot is operating autonomously during testing, it is also recording its GPS location and saving it to a locally stored output file, or to a remotely hosted GIS server if it has a current network connection.

4.4 Results

The results of testing on both paths are presented in Tables 4.2 and 4.3. The trained robot was able to successfully navigate an average of 87% of the total urban path, and an average of 94% of the full park path distance. The trained controllers tend to be very successful (90%+ completion) or fairly unsuccessful (eg. Park Training 3 and Park

	Test	**Percent Driven**
Training 1	1	100
	2	50
	3	100
	4	100
	5	100
	6	100
	Average	92
Training 2	1	100
	2	100
	3	100
	4	86
	5	86
	6	100
	Average	95
Training 3	1	100
	2	79
	3	100
	4	100
	5	86
	6	100
	Average	94
Training 4	1	50
	2	21
	3	64
	4	57
	5	50
	6	71
	Average	52
Training 5	1	100
	2	21
	3	100
	4	100
	5	100
	6	100
	Average	87
Total	Average	87

Table 4.2: Results of testing in the urban environment show the robot was capable of navigating 87 percent of the total length of the path on average.

	Test	**Percent Driven**
Training 1	1	50
	2	67
	3	100
	4	67
	5	75
	6	75
	Average	72
Training 2	1	100
	2	100
	3	100
	4	100
	5	100
	6	100
	Average	100
Training 3	1	33
	2	33
	3	50
	4	50
	5	33
	6	67
	Average	44
Training 4	1	100
	2	100
	3	100
	4	100
	5	100
	6	100
	Average	100
Training 5	1	100
	2	100
	3	100
	4	67
	5	100
	6	100
	Average	94
Total	Average	94

Table 4.3: Results of testing in the park environment show the robot was capable of navigating 94 percent of the total length of the path on average.

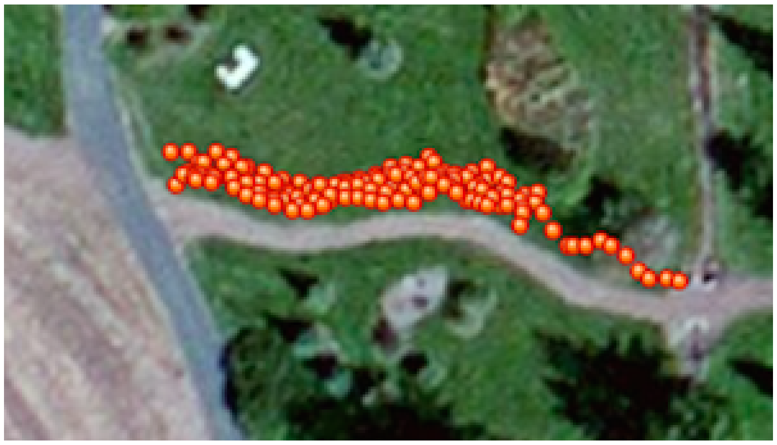


Figure 4.4: This image shows the GPS locations of a robot during autonomous operation on the park path. It's clear that the points are following the general shape and length of the path, although they are not located directly on the path in this image. This offset difference is due to a difference in the map projection used when visualizing the data, and is not due to any inherent defect in the mapping system. More points are located near the beginning of the path (the left side in the image) since over multiple trials the robot generally spends more total time near the beginning as it starts to fail nearer to the end of the path.

Training 4) which shows that this training method was capable of producing successful controllers. Less successful controllers could stem from poor human demonstration, or they were, more likely, from limitations in the robotic vision system and changes in lighting during the training and testing phases. In particular, if human demonstration took place in brightly lit conditions, and testing took place during cloudy conditions, performance may lessen due to changes in what is perceived as safe road. These results show that on-line, on-board evolutionary Learning from Demonstration is an effective method of training robots to complete path following tasks. The method proposed in this work is general enough to successfully train robots in two distinctly different outdoor environments with two different robotic platforms.

In addition, the robot was capable of recording its current GPS position during autonomous operation enabling the mapping of its operational location (see Figure 4.4).

4.5 Proof of Concept Test on Long Trail

As part of testing the presented training method in outdoor environments, a proof of concept experiment was run to examine whether the robot was capable of autonomously navigating an outdoor trail for a much longer distance than anything done in the controlled experiments presented previously.

This experiment was carried out using the wheeled rock crawler platform. The path used for training was a dark gravel path with a variety of different materials on either side including, grass, trees, dirt, and bark. The goal of the experiment was to see how far the robot could navigate without human intervention.

The evolutionary parameters were kept the same as in the previous outdoor experiments. Evolution didn't start until 3 of each training case where given to the robot. The single difference in training parameters was that a road library of size 30 was used rather than of size 10. This larger library was used in order to account for the larger variability in trail conditions the robot was likely to encounter over the larger distance. In particular, the robot was given 10 examples of trail that was in shade to help it recognize the shaded regions on the trail.

When training was complete, the robot was placed in the center of the gravel path facing south, and was allowed to autonomously navigate the path. It traveled a total of .25 miles with a total of two human interventions. It traveled approximately half of this distance without any intervention, and then experienced two regions of path that were highly shaded with very different bordering materials than what it was trained with. The robot was able to successfully navigate through several other regions of shade or unfamiliar surrounding materials, and it was likely a combination of difficult features that led to the robots driving off the path in these two regions. The robot was able to record its GPS position during operation and produce accurate location data that could be used to map the navigated path (see Figure 4.5).

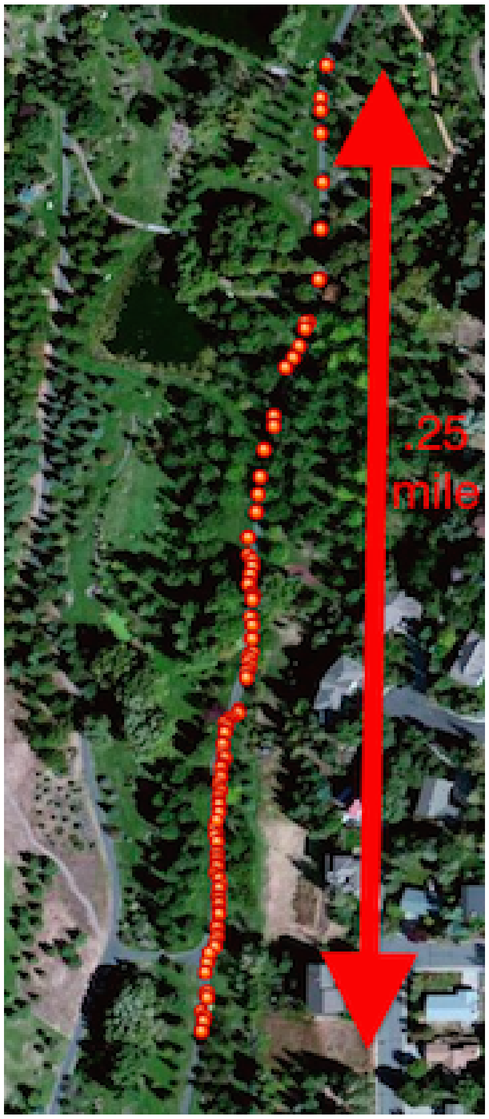


Figure 4.5: This image shows the GPS locations of a robot during autonomous operation on a long gravel path within a park environment. The robot was able to navigate a distance of .25 miles with a total of two human interventions. Both of these interventions simply required placing the robot back in the middle of the path. It's likely the robot lost the path in these situations because the area was extremely shaded, with edges that varied greatly from what it had been originally trained on.

This test shows that the presented approach is capable of working in a much less controlled testing environment, and that it can produce controllers general enough to handle some variation in conditions between training and autonomous operation. The trained controller can still face difficulties when environmental conditions are very different than what it saw during demonstration. This test also demonstrated that the mapping of navigated trails can be done on a larger scale and still produce accurate position data.

4.6 Conclusions

The results show that on-line, on-board evolutionary Learning from Demonstration is a practical approach to training robotic agents. The same algorithm was capable of training robots to complete two different outdoor path following tasks using two different robotic platforms. The results show that this method was capable of producing controllers that could consistently navigate over 87% of the total distance of each path type. In addition to navigating outdoor paths, the robot was capable of mapping its location during operation, which increases the utility of an easily trainable, and inexpensive, autonomous mobile robot.

Chapter 5: Technical Challenges

Through this work, several interesting technical challenges arose. Overcoming these challenges was crucial to the continuation of the research.

5.1 Robot Vision Processing

The work presented in this thesis relied on image processing systems that took input images from a smartphone camera and produced output values for a controlling neural network. These output values represented the probability that an input sub region was a safe, known road type. While this type of system is simple in concept, the implementation of any robotic vision system, is challenging for a number of reasons.

5.1.1 Challenges to Robotic Vision Systems

Robotic vision systems that relay on camera input are a common occurrence in robotics research [58, 2, 31]. These systems typically take some camera input, and generate output values for a robotic controller. A common method of processing is the use of image color to classify image regions. This can be a simple approach in theory, but in practice there are a number of challenges to account for.

Variations in Lighting

The human eye perceives the color of an object as generally being the same regardless of the current illumination conditions [40]. A camera does not operate this way however. The same object in two different images can appear to have very different color if the illumination was different when the images were taken. Additionally, many digital cameras have built in color correction, white balancing, and contrast correction. All of these factors can produce images that may look quite different that what the human eye perceives [30].

These differences pose a significant problem for robotic vision systems that rely on images to classify features as road and non-road. The same colored path may appear very differently to a camera under different lighting conditions, when the camera is held at a different angle, or when different automatic corrections where applied by the camera hardware. This makes classifying that path difficult, and is something that should be accounted for when implementing a camera based vision system.

Blurring of the Camera Image

Another challenge to image processing for mobile robots, is the blurring of camera images while the robot is moving during operation. If the robot is moving fast enough, an input image can be very blurred, making it nearly impossible to accurately detect features. However, even small amounts of blurring can negatively impact how features are recognized by changing the perceived color of an object in the image.

Camera Quality

The quality of the camera must be considered when analyzing a robotic vision system. Modern cameras range greatly in quality and features. The same image processing system could perform differently using different cameras due to changes in image size, the auto corrections performed or camera focus. Additionally, if a system is meant to process images in real time, a larger, higher quality image might not necessarily be better due to a larger number of pixels to process which takes additional computation time.

5.1.2 Initial Image Processing System

For the work presented here, an image processing system was used that took images from a smartphone camera, classified regions of the image as road or non-road, and produced output values for a controlling neural network. This was modeled after the approach taken in [58] and used pixel color as the distinguishing feature for classifying image regions.

Road Library

To classify image regions, a library of known road types is generated at the start of the robotic training process (Algorithm 2 describes the library creation process). When images are processed during operation, this library is used to classify input regions and produce output values. The library is created under the assumption that whatever is visible in the image, directly in front of the robot, at the time the library is being built, is an acceptable road type. To enforce this assumption the library is built by sampling image data only from a small 50px $\times$ 50px region located at the bottom center of the input image (see Figure 5.1a). This size was chosen because it was small enough that when the robot turned, the region didn't leave the road. When image data was taken from the input region, normalized histograms for the red(R), green(G), blue(B), hue(H), and saturation(S) components of the region were used to build a single road model representing an example of a known road type. During initial testing, using features from both the RGB and HSV color space seemed to improve the performance of recognizing road from non-road regions. Normalized histograms were created using the OpenCV[a] functions *calcHist*, and *normalize*.

When each road model is created, it can be stored either as a new example of an existing road type, or as the first example of an entirely new type of road. For example, several models in a row could be considered "shaded path." The robot could then move from the shade into a well lit area and the next model would be the first example of "brightly lit path."

The determination of whether a new model belongs to an existing road type, or is a new type of road was made by a measuring the correlation between the new model,

[a]`http://opencv.org/`

and the models present in the existing road types. Two models' similarity was defined as the average correlation between the RGBHS components of each model. Correlations were calculated with the OpenCV function *compareHist* where an output value of 1.0 means the two histograms are positively correlated, a 0 means no correlation, and -1.0 is a negative correlation between the two histograms. A similarity threshold 0.75 was used to classify two models as similar or not. If the best correlation between the new model, and all existing models, was less than 0.75, the new model was used as an example of a new road type. If the best correlation between the new model, and a model present in an existing road type, was higher than 0.75 the new model was added as a new example of the existing road type.

Only ten unique road types, and ten examples of each road type, could be present in the road library for a total of 100 models. This restriction was set so the number of stored models would have an upper bound, and could not grow indefinitely during operation. This maximum value had to be considered in two situations:

1. The newly created model is an example of a new road type, but the maximum number of road types has already been reached

2. The newly created model is a new example of an existing road type, but the maximum number of examples has already been reached

In the first situation, the least used road type is removed from the road library, and the new road example is added. In the second situation, the least used model of the road type is replaced by the new model. The least used is determined by taking the minimum selection frequency of either a road type or road type model as follows:

- Least Used Road Type: $\frac{Total\, selections\, of\, all\, examples}{Sum\, of\, all\, region\, comparisons}$

- Least Used Road Type Model: $\frac{Total\, selections\, of\, example}{Region\, comparisons}$

A road type example is selected when it is the most similar model to an input region, and a road type is selected when one if its example models is the most similar model to an input region. Through this process, new types of road can be learned and recognized, while older examples of road may be discarded. This assumes that the robot's current environment is more relevant to its operation than older environmental conditions.

Algorithm 2 Road Library Creation

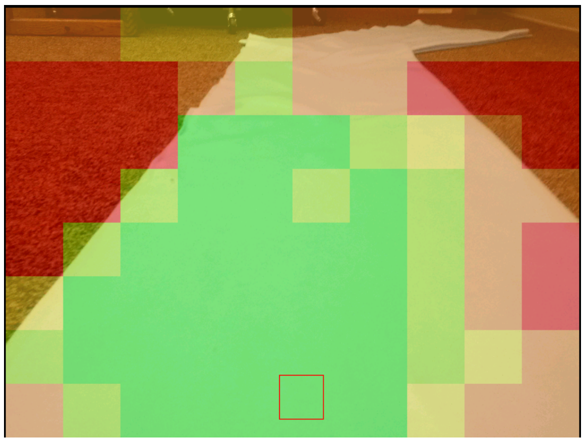
```
generate new road model

{Find best matching model}
for all $roadTypes \in roadLibrary$ do
    for all $model \in roadType$ do
        $correlation = \text{compareHistograms}(newModel, model)$
        if $correlation > bestCorrelation$ then
            $bestCorrelation = correlation$
            $bestRoadType = roadType$
        end if
    end for
end for

{Update or add to the road model list}
if $bestCorrelation > similarityThreshold$ then
    if max number of example already reached then
        remove least used model
    end if
    add new model to matching road type
else
    if max number of road types already reached then
        remove least used road type
    end if
    add new road type using new model
end if
```

Run-time Image Processing

While the robot is operating autonomously, images are processed to produce inputs for the controlling neural network. An image is taken from the camera, split into sub regions, and each sub region is compared to the road library. Every region is assigned a value based on its highest correlation with a model in the road library. The resulting values are then sent to the neural network as inputs. Algorithm 3 describes the image processing procedure.

Algorithm 3 Image Processing

```
for $subregion \in Image$ do
  for all $roadTypes \in roadLibrary$ do
    for all $model \in roadType$ do
      $correlation = compareHistograms(subregion, model)$
      if $correlation > bestCorrelation$ then
        $bestCorrelation = correlation$
      end if
    end for
  end for
  $outputValues.add(bestCorrelation)$
end for
```


(a) Single example in the road library.

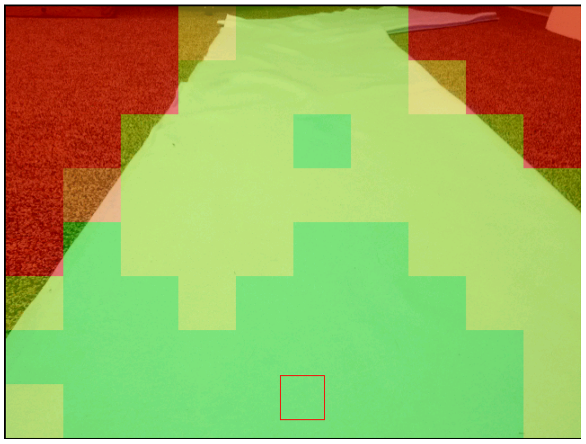

(b) Five examples in the road library.

Figure 5.1: These images show the results from image processing after a single example has been added to the road library, and after several examples of the path have been added to the library. With a larger number of examples in the color library, the classification of road and non-road is much more accurate.

5.1.3 Revised Approach

A revised approach to image processing was developed to simplify processing and increase performance. The new approach differed from the initial approach in several key ways. First, the road library was built during a specific vision step prior to any learning phase. Because of this, the construction of the library was simplified to store a list of road models, rather than grouping the models by similarity. Algorithm 4 describes the revised library creation process. A road model was also simplified to be a single color histogram containing red, green, and blue channels of the input image, rather than several individual histograms as in the initial version.

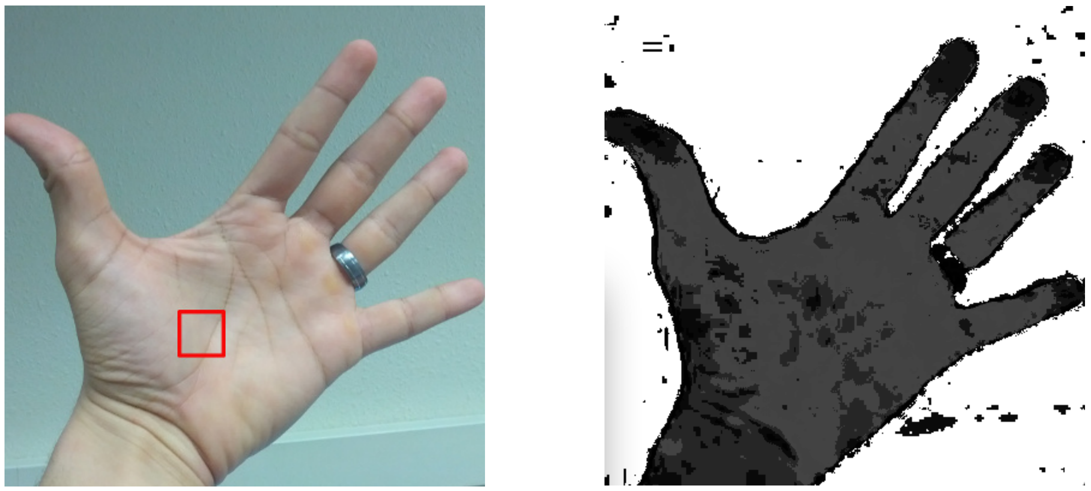
The largest change in the revised approach was the way in which road models were compared. Rather than using the average correlation from all separate histograms, the single 3D histograms were used with the input image in a back projection calculation using the OpenCV function *calcBackProject*. The back projection operation creates a new image where each pixel represents the probability of the same pixel in the input image belonging to the color distribution found in the road model. To generate output values, each sub region in the results back projection image was averaged to generate the average probability of that region belonging to a known road type. Theses probabilities were then given as inputs to the robot's controller.

Algorithm 4 Revised Road Library Creation

```
N = desired road library size
for i = 1 to N do
    generate road model
    add new model to road library
end for
```

Native Processing

Each of the previously described image processing methods were implemented entirely in Java using the OpenCV4Android SDK. The Java only portion of the SDK provides



(a) Input image (b) Processed image.

Figure 5.2: These images demonstrate how the back projection calculation classifies an image. Figure 5.2a is the image that will be processed using backprojection. The red square located within the hand represents the area from which the sample histogram was taken. This is the region that the other pixels will be compared to during the back projection calculation. Figure 5.2b is the image generated from performing a back projection calculation on the input image to the left, using the region taken from the hand as the sample color distribution. The processed image represents the probability of each pixel belonging to the same color distribution as the sample region. In this example, it serves to classify areas that have the same color distribution as the hand in the input image.

Algorithm 5 Revised Image Processing

```
for all $subregions \in Image$ do
    for all $model \in roadLibrary$ do
        calculate back projection using subregion and model
        probability = average value of back projected subregion
        if probability > highestProbablilty then
            highestProbablity = probability
        end if
    end for
    subregion output = highestProbability
end for
```

all the functionality necessary for the work described previously, but does not provide the performance of the native C++ implementation of OpenCV. In order to increase the efficiency of image processing, the revised back projection processing method was also implemented in C++ and called from the Android application using the Java Native Interface (JNI). By calling the native C++ OpenCV functions, performance increased by approximately 22% over the Java implementation of the algorithm.

5.1.4 Additional Work

During the development of each image processing system, work was done to expand the features available for image classification. Specifically, different color spaces, and the use of image texture were examined.

The work presented here has all used some combination of RGB color channels for image processing, but the same processing techniques could work for different color spaces as well such as HSV or YCbCr. HSV is commonly used in computer vision application because it separates color and intensity information making it more robust to lighting changes. Some work was done to incorporate HSV into the current image processing algorithm, but to make comparisons across experiments easier, the common RGB color space was primarily used.

Texture

Texture can be useful measure with which to distinguish features from one another. An example where texture could be useful is a path of short, trimmed grass that runs through a field of much taller grass. The color of each section of grass is likely to be quite similar, but the texture of the grass would likely be quite different.

Work was done to capture differences in textures within an image in hopes of using it as an additional classifier for a robotic vision system. Texture of an image was calculated using Local Binary Patterns (LBP). The resulting texture images were then used to build histograms that were compared using a correlation measure to one another to determine how similar they were to a road library of texture images. This process worked, but the accuracy was not as high as that of color features. This is an avenue for future work. The use of a texture feature with a color feature could have a significant impact on the performance of an image processing system.

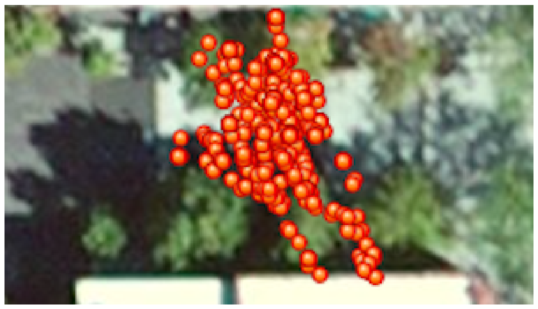
5.2 Locational Awareness and Mapping

While the goal of this research is to present a training method for autonomous mobile robots, any real world application of such a robot requires the ability to track where a robot is and has been. If a robot breaks down, gets lost, or needs to be picked up for any reason, it will be necessary to know where that robot is. In addition, recording the mobile robot's position can have useful applications such as autonomous trail mapping.

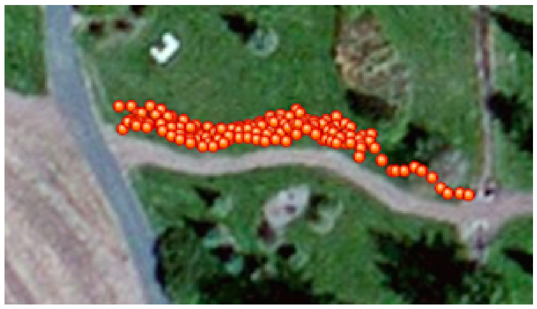
As part of this thesis work, an autonomous mapping module was developed for the robot. This module allows the robot to track its GPS position and save those positions to a csv output file. In addition, if the robot has a network connection, the GPS data can be sent to an ArcGIS feature service[b] hosted in the cloud at ArcGIS.com [c].

[b]ArcGIS feature service: `http://resources.arcgis.com/en/help/main/10.1/index.html#//0154000002w8000000`

[c]ArcGIS.com: `http://www.arcgis.com/features/`



(a) GPS data collected during autonomous operation in an urban environment.



(b) GPS data collected duing autonomous operation in an open, park environment.

Figure 5.3: Images showing GPS data collected during autonomous operation in different environments. The data collected from the urban environment has little resemblance to the actual path that was traversed, while data from the park closely resembles the path that was followed. This difference is due to limitations in GPS technology. In the urban environment there was significant tree canopy overhead and there were buildings on either side of the path. Both of these can limit the accuracy of GPS. The robot is also never in one position for longer than a second which reduces the fine grain accuracy of the recorded position.

Chapter 6: Conclusions

This research has demonstrated an on-line, on-board evolutionary Learning from Demonstration system that is capable of training autonomous mobile robots to navigate both indoor and outdoor paths. This is an important contribution for two key reasons.

The first key contribution is that the work presented here is done entirely in real time, on-board real robotic hardware that is much less expensive, and much more computationally powerful than traditional research robots. This contrasts with the vast majority of evolutionary robotics research, which is typically performed in simulated environments to avoid the difficulties and time constraints of evolving in real world robotic hardware. The second key contribution is the successful combination of Learning from Demonstration with evolutionary learning techniques. This combination of approaches hasn't been commonly used, but this research has shown it to be an effective way to train robots.

To improve the performance of the evolutionary LfD system, considerable research was done into different methods of image processing for robotic vision. Several methods were developed and to classify robotic input images as road or non-road regions. This type of classification was the basis for creating behavior training sets and autonomous operation based on camera input. It was found that the performance of the vision system was crucial to the overall performance of evolved neurocontrollers. Therefore, it was very important to continue improving the image processing system through the course of this research.

In addition to the development of image processing systems, a mapping module was created that could be run on the robot during autonomous operation to enable the mapping of the robot's operational behavior. The ability to track and store its location immediately allows the robot to autonomously map trails or paths that it has been trained to drive, and could have more significant impact in other practical applications such as search and rescue.

All these contributions together lay the groundwork for developing an easy to train, adaptable, affordable, autonomous robot. This type of robot could be applied to a variety of tasks and environments such as extraterrestrial exploration or search and rescue and would have the potential to save time, money, and lives depending on the application.

Chapter 7: Future Work

There are several directions for future work that can build off of what is presented here, all of which could add to the effectiveness of the evolutionary Learning from Demonstration system, and to the usefulness of the robotic agents used.

The neural networks used to control the robot were quite simple, and all that was evolved were the weights. The architecture of the network itself was fixed throughout evolution. Using a more complex representation, such as NEAT, that allows the architecture of the network to evolve could increase the performance of the system, particularly if a more challenging test problem is used. Another change to the controlling neural network could be to make it a recurrent network, and give it some basic memory of its previous actions. This might allow the robot to avoid getting stuck in situations where the current image input alone is not enough to indicate the correct action to take.

Modifying the way outputs are determine from using discrete left, right, forward actions based on the highest output activation to using a continuous vector based motion based on the combined values of the output nodes would allow the robots to make more fine-grained actions that would be better suited to path following tasks. The ability to continue training a robot after evolution is finished would be a good step towards making this type of system even more flexible. If training could continue after observing the robot's performance in autonomous mode, it would allow the retraining of problem behaviors, or could allow for incremental training of complicated tasks. Additionally, the ability to continue training could be combined with some sense of the robot "needing help" when it's unsure of what to do, and could allow for more robust behaviors to be developed.

Further refinements of the current vision system could also improve the performance of the evolutionary LfD system. Possible improvements include:

1. Using an additional road library to recognize a second color or road type could

allow for more complex behavior to be trained

2. The use of a more illumination invariant color space such as HSV could reduce the impact of changes in lighting conditions

3. Implementing some type of illumination correction to match the current environment to the illumination that the library was trained on would reduce the impact of lighting changes

4. Implementing all image processing code in C++ and/or making use of the GPU to carry out image processing calculations could lead to large decreases in processing time

A useful improvement to the mapping module could be to make use of geofencing to limit the operational area of the robot when in autonomous mode. By defining a geographic area that the robot is constrained to ahead of time, the mapping module could track its current location against the geofence and potentially stop operation if it leaves the area, or it could simple be trained to turn around when the robot drives outside the geofence. This type of behavior would be very useful in practical applications where the safety of the robot is important.

Bibliography

[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] JMA Alvarez, Theo Gevers, and Antonio M López. Vision-based road detection using road models. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 2073–2076. IEEE, 2009.

[3] 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.

[4] 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.

[5] 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.

[6] Atta-ul-Qayyum Arif, Dimitar G Nedev, and Evert Haasdijk. Controlling evaluation duration in on-line, on-board evolutionary robotics. In *Evolving and Adaptive Intelligent Systems (EAIS), 2013 IEEE Conference on*, pages 84–90. IEEE, 2013.

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

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

[9] 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.

[10] Nicolas Bredeche, Evert Haasdijk, and AE Eiben. On-line, on-board evolution of robot controllers. In *Artifical Evolution*, pages 110–121. Springer, 2010.

[11] Rodney A Brooks. Artifical life and real robots. In *Toward a practice of autonomous systems: Proc. of the 1st Europ. Conf. on Artificial Life*, page 3, 1992.

[12] Sylvain Calinon, Florent D'halluin, Eric L Sauser, Darwin G Caldwell, and Aude G Billard. Learning and reproduction of gestures by imitation. *Robotics & Automation Magazine, IEEE*, 17(2):44–54, 2010.

[13] 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.

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

[15] Anders Lyhne Christensen and Marco Dorigo. Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In *Proceedings of the 10th International Conference on the Simulation and Synthesis of Living Systems (Alife X)*, pages 248–254. Citeseer, 2006.

[16] Travis DeVault, Nathaniel Ebel, Juan Marulanda, Jayandra Pokharel, Terence Soule, and Robert B. Heckendorn. Neuroevolution of a trail following robotic controller for an autonomous cotsbot. *In Submission*, 2014.

[17] Marco Dorigo. Swarm-bot: An experiment in swarm robotics. In *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pages 192–200. IEEE, 2005.

[18] AE Eiben, Nicolas Bredeche, Mark Hoogendoorn, Juergen Stradner, Jon Timmis, Andy Tyrrell, and Alan Winfield. The triangle of life. In *Advances in Artificial Life, ECAL*, volume 12, pages 1056–1063, 2013.

[19] AE Eiben, Evert Haasdijk, Nicolas Bredeche, et al. Embodied, on-line, on-board evolution for autonomous robotics. *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution.*, 7:361–382, 2010.

[20] Agoston E Eiben and James E Smith. Introduction to evolutionary computing (natural computing series). 2008.

[21] Anna Isabel Esparcia-Alczar, A Martnez-Garca, Antonio Mora, JJ Merelo, and Pablo García-Sánchez. Controlling bots in a first person shooter game using genetic algorithms. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[22] 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.

[23] Dario Floreano, Francesco Mondada, et al. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. *From animals to animats*, 3:421–430, 1994.

[24] James A Foster. Evolutionary computation. *Nature Reviews Genetics*, 2(6):428–436, 2001.

[25] Daniel H Grollman and Odest Chadwicke Jenkins. Learning robot soccer skills from demonstration. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 276–281. IEEE, 2007.

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

[27] 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.

[28] 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.

[29] 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.

[30] Jun Jianga, Yonghui Zhaob, and Shen-ge Wangb. Color correction of smartphone photos with prior knowledge. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 8302, page 10, 2012.

[31] Ming-Yi Ju and Ji-Rong Lee. Vision-based mobile robot navigation using active learning concept. In *Advanced Robotics and Intelligent Systems (ARIS), 2013 International Conference on*, pages 122–129. IEEE, 2013.

[32] 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.

[33] AJ Keane. The design of a satellite beam with enhanced vibration performance using genetic algorithm techniques. *The Journal of the Acoustical Society of America*, 99(4):2599–2603, 1996.

[34] L König, Kristof Jebens, Serge Kernbach, and Paul Levi. Stability of on-line and on-board evolving of adaptive collective behavior. In *European Robotics Symposium 2008*, pages 293–302. Springer, 2008.

[35] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126. ACM, 2010.

[36] Aleš Kubık. Distributed genetic algorithm: A case-study of evolution by direct exchange of chromosomes. *Computing and Informatics*, 23:575–596, 2004.

[37] Aleš Kubík. Distributed genetic algorithm: a case-study of evolution by direct exchange of chromosomes. *Computing and informatics*, 22(6):575–596, 2012.

[38] Juan Luis Jiménez Laredo, AE Eiben, Maarten van Steen, and Juan Julián Merelo. Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246, 2010.

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

[40] Laurence T Maloney and Brian A Wandell. Color constancy: a method for recovering surface spectral reflectance. *JOSA A*, 3(1):29–33, 1986.

[41] Jean-Marc Montanier and Nicolas Bredeche. Embedded evolutionary robotics: The (1+ 1)-restart-online adaptation algorithm. In *New Horizons in Evolutionary Robotics*, pages 155–169. Springer, 2011.

[42] Ulrich Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. 2002.

[43] Cédric Notredame and Desmond G Higgins. Saga: sequence alignment by genetic algorithm. *Nucleic acids research*, 24(8):1515–1524, 1996.

[44] 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.

[45] Fernando Silva, Paulo Urbano, Sancho Oliveira, and Anders Lyhne Christensen. odneat: An algorithm for distributed online, onboard evolution of robot behaviours. In *Artificial Life*, volume 13, pages 251–258, 2012.

[46] 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.

[47] 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.

[48] 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.

[49] Terence Soule and Robert B Heckendorn. A practical platform for on-line genetic programming for robotics. In *Genetic Programming Theory and Practice X*, pages 15–29. Springer, 2013.

[50] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE*, pages 182–189, 2005.

[51] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[52] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[53] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[54] 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.

[55] Keith Sullivan and Sean Luke. Real-time training of team soccer behaviors. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 356–367. Springer, 2013.

[56] 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.

[57] Yukiya Usui Takaya and Takaya Arita. Situated and embodied evolution in collective evolutionary robotics. In *In Proc. of the 8th International Symposium on Artificial Life and Robotics*. Citeseer, 2003.

[58] Ceryen Tan, Tsai Hong, Tommy Chang, and Michael Shneier. Color model-based real-time learning for road following. In *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, pages 939–944. IEEE, 2006.

[59] 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.

[60] Richard A Watson, SG Ficiei, and Jordan B Pollack. Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[61] 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.

[62] Ching Zhang and Andrew KC Wong. Toward efficient multiple molecular sequence alignment: a system of genetic algorithm and dynamic programming. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 27(6):918–932, 1997.