Noname manuscript No.
(will be inserted by the editor)

Learning from Demonstration for Encapsulated Evolution of Robotic Controllers

Travis DeVault, Nathaniel Ebel, Juan Marulanda, Jayandra Pokharel, Terence Soule, Robert B. Heckendorn

Received: date / Accepted: date

1 Abstract

In this paper we present a novel approach to encapsulated evolution using an inexpensive, but computationally powerful, robot and learning from demonstration. Encapsulated evolution, in which a robot maintains an evolving population of behaviors, is a common technique for allowing robots to adapt, in real-time, to changing conditions and environments. Research on encapsulated evolution has been hampered by a lack of inexpensive robots with the computational power to run an evolutionary algorithm. Additionally, encapsulated evolution commonly uses unsupervised learning techniques, which face several challenges in real-world environments. We present two approaches to overcoming these difficulties. First, we present an inexpensive, computationally powerful robot capable of maintaining large populations of complex individuals. Second, we combine encapsulated evolution with learning from demonstration, a supervised learning technique, which avoids many of the problems associated with unsupervised learning in the real world. Our results show that learning from demonstration combined with encapsulated evolution can be carried out in real robotic hardware using low cost, computationally powered robots.

Keywords Evolutionary Robotics · On-Line · On-Board · Encapsulated · Learning from Demonstration · Genetic Algorithm · COTSBots

This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Travis DeVault, Nathaniel Ebel, Juan Marulanda, Jayandra Pokharel
Department of Computer Science,
University of Idaho, Moscow, ID 83844 USA
E-mail: {deva0562, nebel, maru2593, pokh6200}@vandals.uidaho.edu

Terence Soule, Robert B. Heckendorn
Department of Computer Science,
University of Idaho, Moscow, ID 83844 USA
E-mail: {heckendo, tsoule}@uidaho.edu

2 Introduction

The goal of this paper is to present an approach which addresses some of the major problems of encapsulated, on-board, on-line evolution of robotic controllers. To demonstrate the presented solution, we use a cheap, computationally powerful robot, and evolve a population of neurocontrollers to follow a distinctly colored road.

In encapsulated evolution, robots maintain an on-board population of evolving controllers. Encapsulated evolution has great shown great promise in allowing robots to adapt, in real-time, to changing environments or conditions. Successes with encapsulated evolution would represent a significant advance in robots' ability to learn and perform efficiently and autonomously in complex or dynamic environments.

Maintaining an evolving population is computationally expensive, making on-board evolution of robotic controllers difficult, particularly if the population must evolve in real-time and if the population consists of complex controllers. Thus, most research in on-board evolution has been done in simulation [2]. This introduces the simulation to reality problem (also referred to as the reality gap) [12] and does not accurately portray encapsulated evolution's true potential to produce robots that can adapt to the real world. The little research that has been performed with physical robots has either been limited to very simple representations that can be evolved on a small processor [2] or extremely large, expensive robots [17].

A further difficulty with encapsulated evolution is that it has traditionally used unsupervised learning techniques, in which fitness is measured via a time-sharing system. Each individual in the population is placed in temporary control of the robot and receives a fitness value based on how well the robot performed while that individual is in control. A robot would need an on-board critic to judge the fitness of each proposed solution. This can be problematic in novel complex real world scenarios. This introduces three significant problems. First, evaluation of every member of a population requires allowing each individual temporary control of the robot, which is either extremely time intensive or severely limits the size of the population. Second, performance of the robot suffers when a poor individual is in control. Third, in practice this is a very noisy fitness measure as an individual's fitness depends on the state of the robot when it gains control. For example, if the previous individual left the robot in a particular difficult situation, then the current controller's fitness will suffer. Techniques such as racing [7], a recovery period [2], and revaluation [2] have been developed to minimize these drawbacks, but any unsupervised learning technique will suffer from these limitations to some extent.

We use a robot designed using Commodity Off The Shelf (COTS) design principles to overcome the computational requirements of encapsulated evolution. The robot uses a smart phone for processing, giving it significant computational power and memory. Because of this, algorithms with larger population sizes and complex representations can be implemented on-board the robot. The complete design is described in Section 4.1.

To avoid the problems associated with unsupervised learning we use an evolutionary approach based on learning from demonstration. The robots "observe" a human operator and learn to imitate the human's control decisions. Because the correct control decisions are known (assuming that the human operator always,

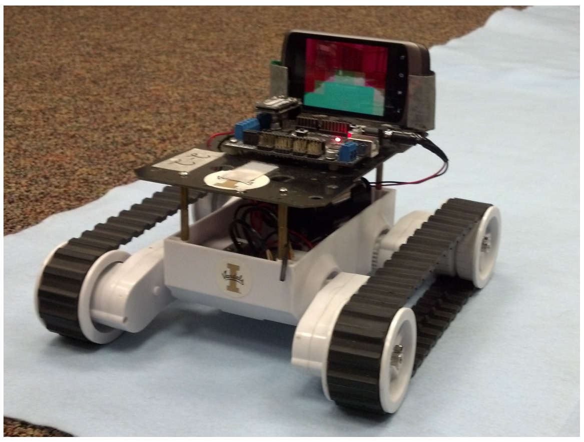

Fig. 1 Robot designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles. The robot consists of three basic components: a smart phone, an Arduino micro controller, and a mobile robot platform. See Section 4.1 for full design details.

or at least usually, makes the right decisions). Individuals can be evaluated based on their ability to replicate the human operator's choices without needing to give them control of the physical robots.

Our results show that our techniques allow for encapsulated evolution of effective robotic controllers. Smart phones were able to perform the evolution of road following neurocontrollers in real time, using encapsulated, on-board evolution.

3 Background

Much of the research into evolutionary robotics has focused on optimizing a controller to allow a single robot to complete a given task during 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 [10]. While this approach works well for some problems, it has several important limitations. Firstly, when evolution is done off-line, the robot does not continue to evolve after the simulation is finished, so it cannot adapt to novel environments. The second limitation is the difficulty in simulating real world environments, particularly when dealing with inputs such as lighting, texture, and sound [4]. Because of this, controllers created during simulation are susceptible to decreases in effectiveness once transferred to the real world [3,12]. This decrease in performance is referred to as the simulation to reality problem. To create controllers that perform well in real-world environments it becomes necessary for controllers to be evolved and tested on a physical robot during real-time operation.

One approach to on-line, on-board evolution is encapsulated evolution in which each robot runs its own evolutionary algorithm on-board, maintaining a population of potential controllers, and testing those controllers on-line during operation. Examples of encapsulated evolution include the (1+1) ON-LINE [2,14] and the (μ + 1) ON-LINE [6,7,9,1] algorithms. These algorithms have been successfully applied to a number of test problems including fast-forward, collective patrolling, and balancing, but testing has only been done in simulation or with small populations of simple controllers.

Previously, encapsulated evolutionary approaches were designed under the assumption that the operating hardware is limited in both processing power and memory. Therefore, the evolutionary algorithms are designed to be "lightweight" using simple representations and small population sizes [2]. In [11, 14] evolution was done on real robotic hardware with processing speeds in the MHz range and memory in the KB range. While these lightweight algorithms have had success in evolving robotic controllers, our work with color following shows that more complex representations are better and therefore other hardware is needed [18]. [18] also demonstrates that smart phones provide powerful hardware that measures processing speeds in the GHz range and has memory in the MB and GB ranges. The availability of this low cost, computationally powerful hardware decreases the need for lightweight algorithms which leads to better performing controllers.

To have robots that can accomplish different tasks with the same controller, robots must be able to learn in some way. Robotic learning falls into two main categories: unsupervised, and supervised.

Unsupervised learning attempts to cluster unlabeled data without any feedback. Common approaches include hidden Markov models and principle component analysis. In evolutionary robotics an unsupervised learning system would have the additional requirement that the robot would need to be able to judge it's own performance, which while easy in simulation, can require additional hardware or software in a physical robot.

Supervised learning strategies involve telling an agent what it should do, given a set of inputs and then using that information to generalize behaviors. An example of this type of learning is using back propagation to train a neural network. Another example of supervised learning is learning from demonstration. In this method a robot learns a task by observing how that task is carried out by an expert. The actual method of learning varies and includes: learning policies, mimicking trajectories, model building, and matching movement primitives [15]. Learning from demonstration has been used to train robots to play soccer [19] and to train a large "crusher" robot to drive through varied, outdoor terrain [17].

Encapsulated evolution is effective in evolving robotic controllers, but previous work has focused on simple representations. It's been shown that more complex representations can lead to better performance. Learning from demonstration has been shown to be an effective method of teaching robots to solve complex problems. Our work combines these methods on a low cost, computationally powerful robot to solve the real world task of trail following.

4 Methods

We use the problem of trail following to show that encapsulated evolution and learning from demonstration can be used together to solve complex problems. This problem was chosen for several reasons. First, the problem of trail following is well suited to learning from demonstration because the desired driving behavior can be easily demonstrated to a robot. Second, the problem is non-trivial and has many real world applications. Third, trail following systems have been done before, but usually on very expensive hardware.

4.1 Test Robot

These experiments were conducted with a physical robot designed and built at the University of Idaho following Commodity Off The Shelf (COTS) design principles. An approach that is becoming more common in robotics research (see for example [8, 13, 18]). The robot consists of three basic components: a smart phone, an Arduino micro controller, and a robot platform.

The smart phone, for these experiments was a HTC Nexus One. This phone has a 1GHz Qualcomm Snapdragon processor, with 512 MB of RAM, and a Qualcomm Adreno 200 GPU.

An Arduino based micro controller with Bluetooth capabilities handles communication between the smart phone and the robot body. For these experiments the DFRomeo micro controller from DFRobot was used.[a] Currently the micro controller is only used to receive commands via bluetooth from the smart phone and to activate the motors on the robotic platform accordingly. Thus, the computational characteristics of the micro controller are not a concern.

A mobile robot platform is needed to carry out the commands given by the smart phone. For these experiments, the Rover 5 Tank chassis was used.[b] This is a small (22.5cm by 24.5cm), tracked, mobile robot platform. It can run continuously for several hours while carrying the micro controller and smart phone.

The smart phone makes decisions and sends output commands to the micro controller via Bluetooth. The micro controller then sends signals to the robot platform's motors to control movement. The robot is completely untethered: it uses the smart phone's computational power to run “full-scale” evolutionary algorithms and the phones built-in sensors as the robot's sensors. For these experiments, the only sensor is the smart phone's camera. During the human operation stage (described below), the smart phone is connected via Bluetooth to a second smart phone that is used as a remote controller.

4.2 Learning Framework

Our evolutionary approach combines learning from demonstration, in which the robot attempts to learn to imitate a human operator, and encapsulated evolution, in which the entire evolutionary process is encapsulated within the robot. The overall process consists of three general stages: human demonstration, learning through evolution, and autonomous operation. We begin by giving a broad overview of the process, followed by a detailed description of each component.

During human demonstration, a human drives the robot, carefully following the desired trail type. The human uses a remote control with discrete forward, left, and right options. While the human is driving, the robot is building two sets of data based on camera input. The first set of data is a library of safe, known road types built from the input images. Details about how the library is built are given in Section 4.2.1. The screen image is then split into 40, 80px $\times$ 96px, regions and these subregions are assigned a road value using the roads library (see Figure 2). The road value is a heuristic used to compare how similar an image region is no

[a] DFRomeo micro controller `http://www.dfrobot.com`

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

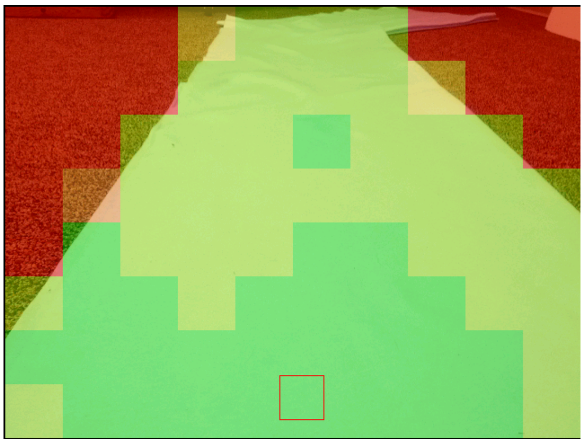

Fig. 2 Robot's screen colored to indicate calculated road values. The box at the bottom is the region from which the road library is built under the assumption that the box never leaves the desired road type.

known, safe road types. These road values, combined with a corresponding action taken by the human, make up the second set of data, which is used for training during the learning stage.

In the learning stage the evolutionary algorithm is used to train a neural network to generate the correct movement command when presented with the trained road/action pairs. The evolutionary algorithm runs until 200 generations have been evolved and evaluated and the individual with the best fitness is then used to control the robot in autonomous mode.

During autonomous operation the robot uses the best evolved neural network to follow a test track autonomously. Input images are continuously compared against the road library to generate road values as inputs for the neural network which then returns its trained action. For these experiments we measure successes and the total number of corners traveled during autonomous operation. A success was defined by:

– The robot driving the entire track (training or testing) while successfully navigating all corners of the track

The robot failed the test if:

– The robot reversed direction on the track
– Both the robot's treads were completely off the track

Because the testing track contained corners of differing angles than the training track, successes on the test track indicate how effective the robot was in learning generalized behaviors that can be applied to different test problems. Section 4.2.5 fully describes the training and testing tracks.

4.2.1 Library of road recognition histograms

To determine what are safe road regions, a library of known road types is generated during human demonstration (Algorithm 1 describes the library creation process).

The library is created under the assumption that whatever is directly in front of the robot is acceptable road. Based on this assumption, a small 50px × 50px region from the bottom center of the input image is used to build the road library (see Figure 2). This size was chosen because it was small enough that when the robot turned, the region didn't leave the road.

As in [20], image histograms were built from the sample region and were used to represent examples of safe road types. Normalized histograms for red, green, blue, hue, and saturation components of an image were used to construct a single road model. 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. The histograms were built using the OpenCV[c] function *calcHist*, normalized using the *normalize* function, and were then stored as a model representing an example of a known road type.

When a model is created, it can either be used as a new example of an existing road type, or as the first example of a new road type. For example, several models in a row could be considered "lit road" and the next model might be the first example of "shaded road." Two models' similarity corresponds to the average correlation between the RGBHS components of each image. Correlation tests were performed 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. A threshold similarity value of 0.75 was used to determine if two models were examples of the same road type. If the similarity of two models was less than 0.75, the new model was used as an example of a new road type.

A limit of 10 unique road types and road type examples was used. This was done so the number of stored models did not grow indefinitely during operation. This maximum value had to be considered in two situations:

1. A new road type has been found, but the maximum number of road types has already been reached
2. A new example of an existing road type has been found, but the maximum number of examples has already been reached

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

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

A road type example is considered selected when it is the most similar model to an input region, and a road type is considered selected when one if its example models is the most similar model to an input region.

4.2.2 Image processing

The goal of the image processing step is to partition an image into regions and to assign each of these regions a value corresponding to how likely it is to contain

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

Algorithm 1 Road Library Creation

```
newModel generated from safe road region

{Find best matching model}
for all $roadTypes \in roadLibrary$ do
  for all $roadExamples \in roadType$ do
    $correlation = \text{compareHistograms}(newModel, roadExample)$
    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 $numberOfRoadTypeExamples() == maxModelSize$ then
    $removeLeastUsedExample()$
  end if
  $bestRoadType.addExample(newModel)$
else
  if $numberOfRoadTypes == maxModelSize$ then
    $removeLeastUsedRoadType()$
  end if
  $roadLibrary.addRoadType(newModel)$
end if
```

road. During operation each image received by the phone's camera is processed and generates a set of road values. These values are either used to build a set of training cases, or as inputs to the neural network network depending on whether the robot is in training or autonomous mode.

When an image is received it is partitioned into an 8 $\times$ 5 grid of subregions. These regions are assigned a road value corresponding to the highest correlation between the sub region and road library models. This creates 40 road values (one for each of the sub regions in the image) which are used to build training cases with an associated output action, or as the inputs to the controlling neural network. Algorithm 2 describes the image processing step.

4.2.3 Neural network architecture

The neural network has forty input nodes, a single hidden layer consisting of five nodes, and an output layer with three nodes. All of the nodes in the network use a sigmoid activation function. The input layer corresponds to the forty 80px $\times$ 96px regions in the partitioned image, one input per region. Each region is evaluated against the constructed histograms of road models. A histogram is constructed from the input region, and then it's correlation with each color model in the roads library is calculated. The highest correlation is used as the input value for each

Algorithm 2 Image Processing

```
for $subRegion \in Image$ do
  for all $roadTypes \in roadLibrary$ do
    for all $roadExamples \in roadType$ do
      $correlation = average\, RGBHS\, correlation$
      if $correlation > bestCorrelation$ then
        $bestCorrelation = correlation$
      end if
    end for
  end for
  $outputValues.add(bestCorrelation)$
end for
```

region to the neural network. The three output neurons correspond to the three discrete actions forward, left, and right. The action is determined by the output neuron with the highest activation level. This is a common approach to evolving ANN controllers for autonomous robots as demonstrated in [4]. In cases of a tie, no command is sent to the robot. This would be a very extreme edge case and never occurred during our experiment.

4.2.4 The evolutionary algorithm

An important goal of this project is to test the feasibility of using smart phones for encapsulated evolutionary learning from demonstration. For these initial investigations we intentionally chose to use a simple, but general, evolutionary model. It is likely that a more sophisticated evolutionary model, in particular one developed for the evolution of neural networks such as NEAT [16] or hyperNEAT [5], would yield better results, and will be tried in future work.

We used a standard generational genetic algorithm (GA) to evolve the neural network weights. Individuals are represented as a complete multilayer perceptron neural network; each individual has 215 weights(40 input nodes to 5 hidden nodes = 200, and 5 hidden nodes to 3 output nodes = 15). Initially the weights were randomly set in the range -1.0 to 1.0. Initially each individual's fitness is assigned to 0.0, and is only calculated with the construction of the next generation, this means that the first generation uses a random selection.

Selection was a tournament of size three. Two individuals are selected as parents and produce only one offspring which receives crossover and mutation before being added to the next generation. Crossover was done as a uniform crossover on every set of parents when constructing the offspring. Each offspring will have 215 weights which will be mutated at a rate of .2, on average resulting in 43 mutations per offspring. The amount of mutation was uniformly distributed between -0.1 to 0.1.

Fitness was determined by comparing the cases collected in the training data and the output that the trainer gave to the robot to the actions taken by each individual network with the same training case. The training cases are divided into category based expected action(Left, Right, Forward). The network can earn an equal amount of fitness for getting all the cases in a category correct, regardless

Table 1 Summary of the evolutionary algorithm parameters.

Population size	40
Generations	200
Mutation rate	.2
Crossover rate	100% Uniform
Selection method	Tournament(size 3)

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. Algorithm 3 describes the fitness function which assigns a better fitness to the network if the network's output at the expected winning action is higher than it's output at it's other actions. The difference between the value corresponding to the expected action and the other actions was attempted to be wide, wide being defined as a threshold of 0.4. No testing was done on this threshold value, this value was chosen because it was a mid range value. High and low values were expected to work poorly in the fitness function. If the network chooses the correct action, it always gets a positive fitness for that training case, but the fitness will increment more if the if the difference is at least equal to the threshold.

Algorithm 3 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
```

The other parameters of the evolutionary algorithm are summarized in Table 1.

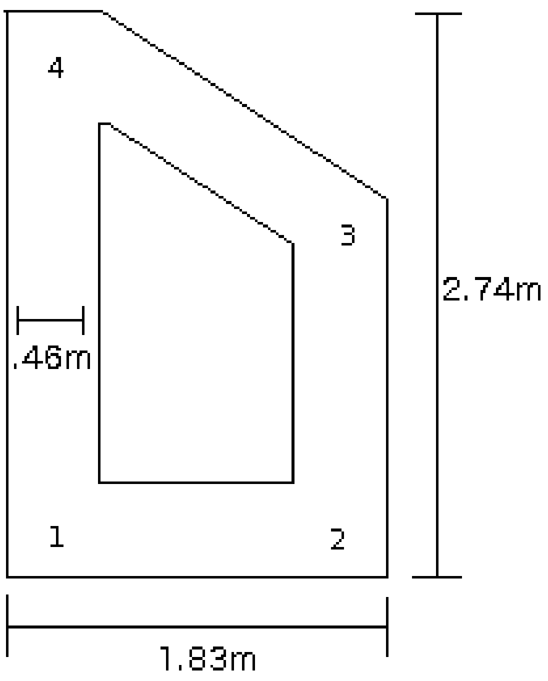

Fig. 3 Track that the robots were tested on. This track is more difficult than the training track because it contains sharper turns than the training track. The numbers at each corner are identifiers used to describe how far around the track the robot was able to navigate during testing.

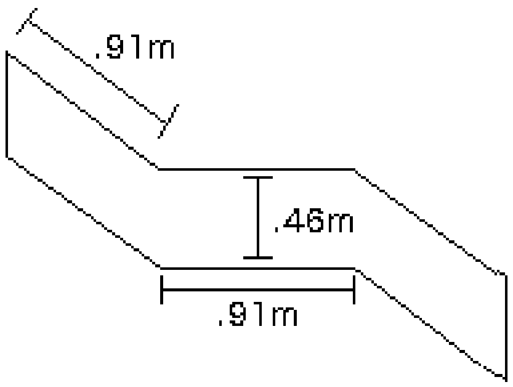

Fig. 4 This track is used to train the robot. The robot is trained by a human going from the right hand side to the left hand side and back two times.

4.2.5 Test conditions and environment

The experimental procedure consisted of repeatedly training a robot on a simple track, using the data collected during training to evolve a neural network, and then testing the evolved neural network by letting it autonomously drive the robot on both the training and testing tracks. The experiments were performed indoors in a well lit room; the tracks were a light blue felt fabric track that contrasted well with the dark brown carpet in the room. The shape of the two tracks are shown in Figure 3 and Figure 4. The training track consisted of two 45 degree turns and a 1.83m straightaway. The testing track was designed to be more difficult and to test the evolved neural networks' ability to generalize. It consisted of two 90 degree turns, one 45 degree turn, and one 135 degree turn.

At the start of human demonstration the robot was given 10 drive commands that are used to build an initial library of road models. With an initial library built at the start of training, initial training cases more closely resemble later ones and reduce the overall noise in the training data. After the initial road library is created, the robot was then driven along the training track down and back two times, although the exact number of given commands depends on the human operator and the specific training. On average this training is sufficient to generate a library of 100 histograms for road recognition and 120-160 road value/command pairs to use during the learning process. At the start of training, and after each command, the robot receives a new image, processes it, generates network inputs, and then draws the image to the screen coloring each subregion based on its assigned road value. The robot then waits to receive a drive command from the human operator. When a command is received the specified action is saved along with the image processing outputs as a training case for ANN training. Once training was completed, the operator started the evolutionary process which generated a neural network controller for the robot. This controller was then used to test the robot during the autonomous stage of operation.

Note that since the evolution happens after the demonstration, and before the robot operates in autonomous mode. This model should be consider off-line learning, because the robot is not learning during operation. The robot has no one of receiving feedback about it's performance during run time. We plan on modifying the evolution to happen during the demonstration phase and during run time to create an on-line learning system in the future.

During autonomous operation the robot attempted to navigate the testing track three times in the clockwise direction, and three times in the counterclockwise direction, for a total of six tests. Each test started from corner 4 because it was perceived as the most difficult, which allowed the robot to attempt as many corners as possible before reaching the most challenging one.

5 Results

Table 2 shows the results of the robot on the training track. The robot was able to successfully navigate the training track 95% of the time indicating that it was successful in learning from demonstration. Table 3 shows the results of the robot on the testing track. Results for the robot in the clockwise and counter clockwise directions are split, showing that on 14 trials in each direction the robot was able to navigate every corner on the track for a total of 28 successes in 60 trials. Note that on corner 3, the number of failures for both clockwise and counterclockwise are 0. This is because corner 3 is the same type of corner that the training track uses. Corners 1, 2, and 4 were all larger angle corners than those the robot had

Table 2 Results of testing the robot on the training track showed the robot could successfully navigate the trail it was trained on 95% of the time.

	Direction 1		Direction 2	
	Corner		Corner	
	1	2	2	1
Times reached	30	29	30	28
Failures	1	0	2	0

Table 3 Number of failures in each direction (clockwise and counterclockwise) on the training track by corner number (see Figure 3). Failures are progressive, if the robot fails to negotiate the N^{th} corner on a test, it doesn't get to the next corner on that test. E.g. when going clockwise the robot always negotiated the first corner (*Corner 3*). It failed to negotiate the second corner (*Corner 2*) 5 of the 21 times it reached it, and failed to negotiate the third corner 1 of the 16 times it reached it. Note that the robot never fails on corner 3 (45 degrees), which is the same angle as the training turn. Failures only occur on sharper, i.e. more difficult, turns.

	Clockwise				Counterclockwise			
	Corner				Corner			
	3	2	1	4	1	2	3	4
Times reached	30	30	23	19	30	27	20	20
Failures	0	7	4	5	3	7	0	6

been trained on. The ability to navigate these corners show an ability of the robot to generalize trained behaviors to novel inputs during operation.

6 Conclusion

While the presented approach showed success in testing, there are several observed limitations that affect the performance of the robot. Inputs from the camera and outputs from image processing have a large impact on robot behavior. The HTC Nexus One phone we used has a narrow degree viewing, so it can be difficult for the camera input to see a severe upcoming turn. The vertical angle at which the phone is placed is also important. Lighting changes significantly depending on the angle of the phone as does the distance in front of the robot the camera can see. These factors can affect both the ability to successfully match road models and the ability to perceive changes to the path. We chose to angle the phone forward at a 30 degree angle which allows the camera to see approximately six feet in front of it and places the "safe road" box 10 inches in front of the robot. We suspect that a greater field of vision would allow the robot to better perceive larger degree corners and could lead to better performance, but further testing with a fish-eye lens is needed.

Another limitation to the current approach is the maneuverability of the robotic platform. The tested Rover 5 platform makes sharp, angled turns that work well for angled corners, but are less desirable for curved tracks. Another platform that was used in preliminary tests had 4 wheels and therefore made more rounded turns.[d] These turns, and a faster operating speed, allowed this platform to quickly

[d] `http://www.nitrorcx.com/51c871-maxstone16-green-24ghz.html`

traverse test tracks, but its limited turning radius made it very difficult to keep the "safe road" box on the track. Keeping the box within the bounds of the track allowed operation under the assumption that whatever is directly in front of the robot is the desired road type. This works well for detecting safe road regions, but it increases the importance of careful training. If the box leaves the desired road surface, it can have a debilitating negative impact on the performance of the image processing. It was for this reason we chose to use the more precise turning Rover 5 as our robotic platform for this research. Because it is slower and has a smaller turning radius, the Rover 5 makes it easier to keep the "safe road" box on the track and therefore simplifies training.

It is likely that a more robust approach can be created by merging these steps. For example, the learning algorithm can be run continuously in the background during human operation. (Human operators tend to send 1-4 control signals per second, this leaves 100's of milliseconds for training between commands, plenty of time for a smart phone's processor or processors to progress on an evolutionary algorithm.) This would allow the robot to take over as soon as its evolving controllers were sufficiently reliable at matching the human operator's decisions. Similarly, if the robot seems to be performing poorly or moves into a novel environment where its road recognition library is no longer appropriate then the human can resume control and the learning process can be continued.

References

1. Arif, A.u.Q., Nedev, D.G., Haasdijk, E.: Controlling evaluation duration in on-line, on-board evolutionary robotics. In: Evolving and Adaptive Intelligent Systems (EAIS), 2013 IEEE Conference on, pp. 84–90. IEEE (2013)
2. Bredeche, N., Haasdijk, E., Eiben, A.: On-line, on-board evolution of robot controllers. In: Artifical Evolution, pp. 110–121. Springer (2010)
3. Brooks, R.A.: Artifical life and real robots. In: Toward a practice of autonomous systems: Proc. of the 1st Europ. Conf. on Artificial Life, p. 3 (1992)
4. Capi, G., Toda, H.: Evolution of neural controllers for robot navigation in human environments. Journal of Computer Science **6**(8), 837 (2010)
5. Clune, J., Beckmann, B.E., Ofria, C., Pennock, R.T.: Evolving coordinated quadruped gaits with the hyperneat generative encoding. In: Evolutionary Computation, 2009. CEC'09. IEEE Congress on, pp. 2764–2771. IEEE (2009)
6. Eiben, A., Haasdijk, E., Bredeche, N., et al.: Embodied, on-line, on-board evolution for autonomous robotics. Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution. **7**, 361–382 (2010)
7. Haasdijk, E., Atta-ul Qayyum, A., Eiben, A.E.: Racing to improve on-line, on-board evolutionary robotics. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 187–194. ACM (2011)
8. Hafiz, A.R., Murase, K.: irov: A robot platform for active vision research and as education tool. In: Advances in Autonomous Mini Robots, pp. 173–181. Springer (2012)
9. Huijsman, R.J., Haasdijk, E., Eiben, A.: An on-line on-board distributed algorithm for evolutionary robotics. In: Artificial Evolution, pp. 73–84. Springer (2012)
10. Karafotias, G., Haasdijk, E., Eiben, Á.E., Haasdijk, E., Eiben, A., Winfield, A., Haasdijk, E., Rusu, A., Eiben, A., Eiben, A., et al.: An algorithm for distributed on-line, on-board evolutionary robotics. In: GECCO, pp. 171–178 (2011)
11. Kernbach, S., Meister, E., Scholz, O., Humza, R., Liedke, J., Ricotti, L., Jemai, J., Havlik, J., Liu, W.: Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution. In: Evolutionary Computation, 2009. CEC'09. IEEE Congress on, pp. 1079–1086. IEEE (2009)
12. Koos, S., Mouret, J.B., Doncieux, S.: Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation, pp. 119–126. ACM (2010)

13. Li, G.: Asccbot: An open mobile robot platform. Ph.D. thesis, Oklahoma State University (2011)
14. Montanier, J.M., Bredeche, N.: Embedded evolutionary robotics: The (1+ 1)-restart-online adaptation algorithm. In: New Horizons in Evolutionary Robotics, pp. 155–169. Springer (2011)
15. Schaal, S., Ijspeert, A., Billard, A.: Computational approaches to motor learning by imitation. Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences **358**(1431), 537–547 (2003)
16. Silva, F., Urbano, P., Oliveira, S., Christensen, A.L.: odneat: An algorithm for distributed online, onboard evolution of robot behaviours. In: Artificial Life, vol. 13, pp. 251–258 (2012)
17. Silver, D., Bagnell, J.A., Stentz, A.: Applied imitation learning for autonomous navigation in complex natural terrain. In: Field and Service Robotics, pp. 249–259. Springer (2010)
18. Soule, T., Heckendorn, R.B.: A practical platform for on-line genetic programming for robotics. In: Genetic Programming Theory and Practice X, pp. 15–29. Springer (2013)
19. Sullivan, K., Luke, S.: Real-time training of team soccer behaviors. In: RoboCup 2012: Robot Soccer World Cup XVI, pp. 356–367. Springer (2013)
20. Tan, C., Hong, T., Chang, T., Shneier, M.: Color model-based real-time learning for road following. In: Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE, pp. 939–944. IEEE (2006)