Neuroevolution of a trail following robotic controller for an autonomous CotsBot

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

Abstract

In this paper we present a novel approach to encapsulated on-board evolution using an inexpensive, but computationally powerful, robotic platform and learning by demonstration. Encapsulated evolution, in which a robot maintains an evolving population of behaviors, is a leading technique for allowing robots to adapt, in real-time, to changing conditions and environments. However, 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 numerous difficulties in real-world environments. In this paper 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 by imitation, a supervised learning technique, which avoids many of the problems associated with unsupervised learning in the real world.

Our results show [...]

1 Introduction

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

Unfortunately, maintaining an evolving population on-board a robot is computationally expensive, particularly if the population must evolve in real-time and if the

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.

population consists of complex controllers. Thus, most research in encapsulated evolution has been done in simulation. This introduces simulation to reality problems (also referred to as the reality gap) [10] 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], extremely large, expensive robots [**?**], or both [].

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. 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 severly 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] (described in detail below) have been developed to minimize these drawbacks, but any unsupervised learning technique will suffer from these limitations to some extent.

To overcome the computational requirements of encapsulated evolution we use a robot designed using Commodity Off The Shelf (COTS) design principles. The robot uses a smart phone for processing, giving it significant computational power and memory. The complete design is described in Section 3.1.

To avoid the problems associated with unsupervised learning we use an evolutionary approach based on learning by 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, 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 [...]

2 Background

To effectivley discuss different implementations of evolutionary robotics it is helpful to use a general classification scheme. Several classification systems have been proposed in previous works [14, 6]. The system presented in [6] classifies embodied evolution, where fitness evaluations are performed on a physical robot, based on where the evolutionary processes take place.

Along with a new method of classification, [6] also presents a vocabulary with which to effectively describe evolutionary robotics. The term "embodied" refers to the physical testing of fitness with 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)

Much of the research into evolutionary robotics has focused on optimzing 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 simluation, and the best individual solution is transfered to a physical robot [9]. 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 evovle after the simulation is finished so it cannot adapt to changing 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 transfered to the real world [3, 10]. This decrease in performance is referred to as the sim-to-reality problem. To create controllers that perform well in real-world environments it becomes neccessary for controllers to be evolved and tested on a physical robot during real-time operation. This becomes increasingly important with highly complex and dynamic environments that cannot be accurately simulated.

One on-line, on-board approach 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. One of the first proposed methods for encapsulated evolution is the (1+1)-ONLINE algorithm which is based on the (1+1) evolutionary algorithm [2, 11]. This work was extended to a $(\mu + 1)$-ONLINE algorithm [6, 7, 8, 1]. These approaches were designed under the assumption that the operating hardware is limited both in processing power and memory. Therefore, the evolutionary algorithms are designed to be "lightweight" and to operate on small population sizes. While they have been successfully applied to a number of test problems including fast-forward, collective patrolling, and balancing, more powerful hardware and algorithms could provide better performance in real-world applications.

While these existing approaches demonstrate effective implementations of encapsulated evolution, actual testing was done in simulation. To more realistically test the effectivness of encapsulated, on-line, on-board evolution it must be tested on actual robots during real operation.

****** Still need to add the following information *******

- Our work with color following showing that more complex representations are better and therefor other hard ware is needed.

- Other examples of current hardware (e.g. Kernback et at "Evolutionary robotics: the next-generation-platform...") showing CPU weaknesses
- How can improved hardware such as our CotsBots increase the effectiness of evolutionary approaches.
- Papers citing problems with unsupervised learning in the real world.
- Papers (review paper) on learning by imitation/demonstration
  - Overview of learning my imitation [**?**]
  - Imitation learning done with the crusher platform[**?**]
  - Learning by demonstration for soccer behaviors [**?**]

Learning by imitation shows great potential in applications of robotic learning and evolutionary robotics. Current evolutionary approaches are designed to operate with limited resources and may not provide enough computational power to solve difficult real world problems. Our work demonstrates that learning by imitation can be used to evolve robotic controllers on a low cost, computationally powerful robotic platform for the real world task of trail following.

3 Methods

3.1 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 [**?**, **?**]). The robot consists of three basic components:

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

The Spinal Cord – an Arduino based microcontroller with Bluetooth capabilities which handles communication between the smart phone and the robot body. For these experiments the DFRomeo microcontroller from DFRobot was used. [1] Currently the microcontroller 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 microcontroller are not a concern.

The Body – a mobile robot platform, for these experiments the Rover 5 thank chasis was used.[2] This is a small (22.5cm by 24.5cm), tracked, mobile robot platform.

[1] `http://www.dfrobot.com/index.php?route=product/product&filter_name=Romeo&product_id=56#.UltL4BaHrzI`

[2] `http://www.dfrobot.com/index.php?route=product/product&path=37_111&product_id=386#.UltPFBaHrzI`

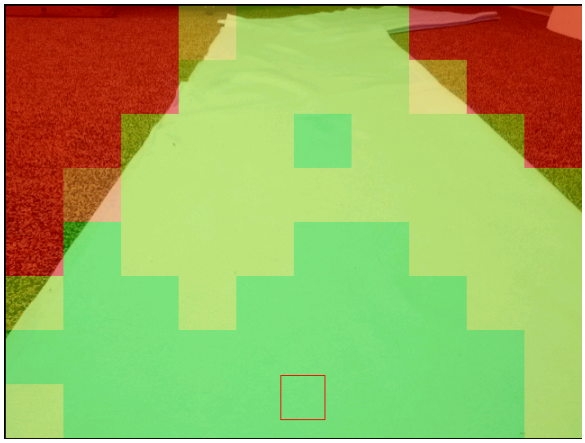

Figure 1: Robot's screen colored to indicate calculated road probabilities. The box at the bottom is the region from which the road library is built under the assumption that the box never leaves the desired road type.

It can run continuously for several hours while carrying the microcontroller and smart phone.

Basically, the brains control the robot, the body moves the robot, and the spinal cord transmits messages between the two via Bluetooth. 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 only the camera is used as a sensor. 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.

3.2 Learning Framework

Our evolutionary approach combines learning by imitation, 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 operation, learning, and autonomous operation. We begin by giving a broad overview of the process, followed by a detailed description of each component.

During human operation 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. First, a library of safe, known road types is built from the input images. Details about how the library is built are given in Section **??**. The screen image is then split into 40, 60px by 96px, regions and these subregions are used to calculate road probabilities using the roads library (see Figure 1). These road probabilities combined with a corresponding action 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 the trained probability/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 continue to follow a test track autonomously. Input images are continuously compared against the road library to generate road probabilities 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. Successes on the test track indicate how effective the robot was in learning generalized behaviors that can be applied to different test problems.

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 miliseconds 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.

In order to generate comparable results between trials, these experiments were conducted with only a single iteration of the overall learning process, in a single environment.

3.2.1 Library of road recognition histograms

To determine the probability that a section of image shows safe road, a library of known road types is generated during human operation under the assumption that whatever is directly in front of the robot is acceptable road. Based on this assumption a small 50px by 50px region from the bottom center of the input image is used to build the road library (see Figure 1). As in [13], image histograms were built from the sample region and stored as models of the road. Normalized histograms for red, green, blue, hue, and saturation components of the image were used as the road models. 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 function *calcHist*, normalized using the *normalize* function, and were then stored as a ColorModel representing an example of a known road type.

When a ColorModel is created, it can either be used as a new example of an existing road type (RoadModel), or it can be used to create an entirely new RoadModel. A RoadModels is defined by a list of similar ColorModels. As new ColorModels are generated and added to a RoadModel, the number of stored ColorModels in the list may reach a max size threshold, in which case the ColorModel that has been selected as the best

match the lowest percentage of the time is removed before the new ColorModel is added. This process is controlled by a histogram similarity threshold which for our experiments was set to 0.75. If the highest correlation between the new ColorModel and all existing ColorModels is less than the similarity threshold, the colorModel is considered to be a new road type and is used to create a new RoadModel. If the similarity is above the threshold, the ColorModel is added to the RoadModel with the highest correlation as a new example of that road type. The RodeModels list uses the same max size threshold as the ColorModel lists and if its size has reached this threshold the RoadModel that has contained the best matching histogram the lowest percentage of the time is removed before the new one is added. Two histograms' similarity corresponds to the average correlation between the two images from each RGBHSV component. Correlation tests were performed with the OpenCV function *compareHist* where an output value of 1.0 means the two histograms are perfectly correlated and the values decrease towards -1.0. Algorithm 1 describes the library creation process.

Algorithm 1 Road Library Creation

```
inputImage = getCameraImage()
safeRegion = inputImage.subRegion()
newColorModel = calculateHistograms(safeRegion)
{Find the best matching, exiting road model}
for all existingRoadModel ∈ roadModelList do
   for all existingColorModel ∈ colorModelList do
      correlation = compareHistograms(newColorModel, existingColorModel)
      if correlation > bestCorrelation then
         bestCorrelation = correlationAvg
         bestRoadModel = existingRoadModel
      end if
   end for
end for
{Update or add to the road model list}
if bestCorrelation > similarityThreshold then
   if bestRoadModel.size() == maxModelSize then
      bestRoadModel.removeLeastUsedColorModel()
   end if
   bestRoadModel.addColorModel(newColorModel)
else
   newRoadModel = createNewRoadModel()
   newRoadModel.addColorModel(newColorModel)
   if roadModelList.size() == maxModelSize then
      roadModelList.removeLeastUsedModel()
   end if
   roadModelList.addRoadModel(newRoadModel)
end if
```

3.2.2 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 road. Thus, during autonomous operation each image received by the phone's camera is processed and generates a set of inputs for the neural network, which decides the robot's next action.

When an image is received it is used to update the road library if the robot is in training mode. Then, regardless of the operation mode, the entire image is partitioned into an 8 x 5 grid of sub regions that are assigned a correlation value corresponding to the highest correlation between the sub region's RGBHS histograms and the histograms stored in the road library. This creates an 8 x 5 array of road probabilities (one for each of the sub regions in the image) which is used as the inputs to the controlling neural network. Algorithm 2 describes the image processing step.

Algorithm 2 Image Processing

```
inputImage = getCameraImage()
for i = 0 to imageWidth by columnWidth do
  for j = 0 to imageHeight by rowHeight do
    subImage(j, x) = inputImage.submat(j, x)
    bestCorrelation = 0
    for all existingRoadModels ∈ roadModelList do
      for all colorModels ∈ colorModelList do
        correlationSum =0
        for all feature ∈ RGBHSVFeatureSpace do
          correlation+ = compareHist(subImage.feature, colorModel.feature)
        end for
        correlationAvg = correlationSum/5
        if correlationAvg > bestCorrelation then
          bestCorrelation = correlationAvg
        end if
      end for
    end for
    outputProbabilitiesList.add(bestCorrelation)
  end for
end for
```

3.2.3 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 by 60px regions in the partitioned image, one input per region. Each region is evaluated against the constructed histograms of road models. A histogram is constucted input region and then it's correlation with each of the color histograms in the road models library is

Population size	40
Generations	200
Mutation rate	.2
Crossover rate	Uniform at .5
Selection method	Tournament(size 3)

Table 1: Summary of the evolutionary algorithm parameters.

calculated. The highest correlation is used as the input value for each region. 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 occured during our experiment.

3.2.4 The evolutionary algorithm

An important goal of this project is to test the feasibility of using smart phones for encapsulated evolutionary learning by imitation. Thus, to determine the generality of the approach 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 [12] or hyperNEAT [5], would yield better results, but it would be less clear that the results were applicable to a broad range of evolutionary techniques.

We used a standard generational genetic algorithm (GA) to evolve the neural network weights. Individuals were represented by an array of [N] doubles corresponding to the [N] weights in the complete neural network. Initially the weights were randomly set in the range [X to Y].

Selection was by...

Crossover was ...

Mutation was ...

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

3.2.5 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 2). The training track consisted of two 45 degree turns and a 6' straightaway. The testing track was designed to be more difficult and to test the evolved neural networks' ability to

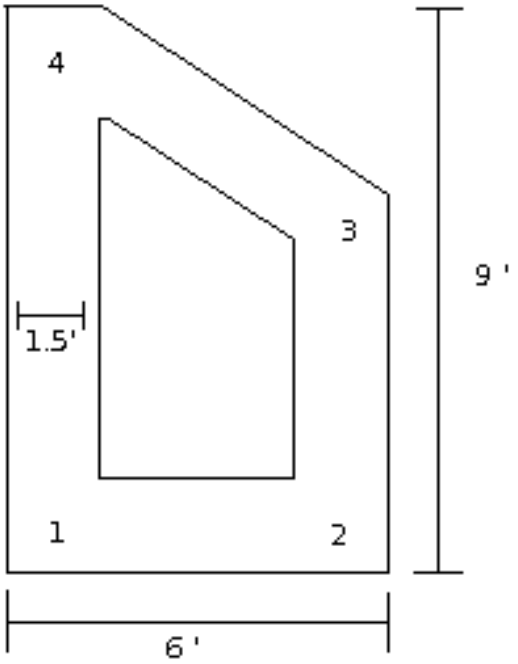

Figure 2: Track that the robots were tested on. This track is more difficult than the training track because it contains sharper turns than the training track. The numbers at each corner are identifiers used to describe how far around the track the robot was able to navigate during testing (see Table **??**).

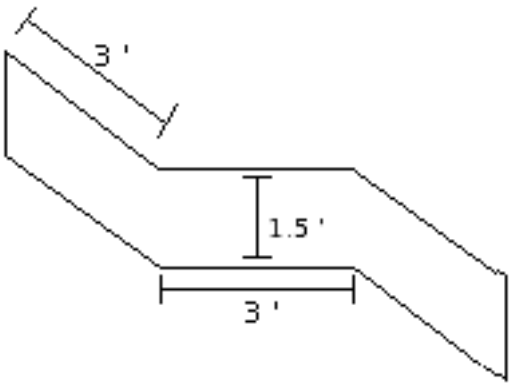

Figure 3: This track is used to train the robot. The robot is trained by a human going from the right hand side to the left hand side and back two times.

generalize. It consisted of two 90 degree turns, one 45 degree turn, and one 135 degree turn.

Clockwise				Counterclockwise			
Corner				Corner			
3	2	1	4	1	2	3	4
21	16	15	12	20	15	15	9

Table 2: Number of successes out of 21 tests in each direction (clockwise and counterclockwise) on the training track by corner number (see Figure 2). 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 negotiated the first corner (*Corner 3*) 21 out of 21 tests. It negotiated the second corner (*Corner 2*) 16 out of 21 tests and the third corner 15 out of *16* attempts.

At the start of human operation the robot was given 10 drive commands that are used soley to build an initial library of RoadModels. Without this step, the road library is empty at the start of the first training and doesn't recognize the road as well as it does after training. This leads to noisy training cases for the ANN. 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. 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 specifc training. On average this training is sufficient to generate a library of 100 histograms for road recognition and 120-160 image/command pairs to using during learning. 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 calculated probabilty of being road. 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.

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.

4 Results

5 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

	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

Table 3: Number of failures in each direction (clockwise and counterclockwise) on the training track by corner number (see Figure 2). 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.

from image processing have a large impact on robot behavior. The HTC Nexus One phone we used only has a 45 degree viewing angle so it can be difficult for the camera input to see a severe upcomming 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 lense is needed.

Another limitation to the current approach is the manuverability 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.[3] These turns, and a faster operating speed, allowed this platform to quickly 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 assmption 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.

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

References

[1] 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.

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

[3] 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.

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

[5] Jeff Clune, Benjamin E Beckmann, Charles Ofria, and Robert T Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE, 2009.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.