Visual Obstacle Avoidance Using Genetic Programming: First Results

Martin C. Martin
Carnegie Mellon University
mm@cmu.edu

Abstract

Genetic Programming is used to create a reactive obstacle avoidance system for an autonomous mobile robot. The evolved programs take a black and white camera image as input and estimate the location of the lowest non-ground pixel in a given column. Traditional computer vision operators such as Sobel gradient magnitude, median filters and the Moravec interest operator are combined arbitrarily. Five memory locations can also be read or written to. The first evolved program is now controlling the robot.

When constructing a system, engineers typically practice iterative design, namely instantiating a design, evaluating it, and then modifying it in light of the evaluation. In the current work Genetic Programming can be seen as automating this process by iteratively improve the architecture of the system in fundamental, previously unplanned ways. The system described here successfully navigates in the hallways outside the lab.

1. Introduction

Computer vision in unstructured environments, such as a typical office environment, is notoriously difficult. Different methods have their strengths and weaknesses, and no one method is universally better or worse than the alternatives. To find a method that works empirically for a particular environment, I use Genetic Programming to evolve arbitrary expressions that combine the results of traditional computer vision operators over a window. The expressions return an estimate of the distance to the nearest obstacle in a given direction. The expression is evaluated for six different directions, and a separate hand-written program uses these estimates to steer the robot.

As in any field of research, one can find threads in the literature by following the evolution of a single idea. The idea that most inspires this work started with Ian Horswill's Ph.D. thesis on Polly the Robot. Polly gave simple tours of the seventh floor of the MIT AI lab, which had a textureless carpeted floor. Obstacles, or at least their boundaries, could therefore be detected as areas of visual texture. The system had problems with other carpet patterns, or even sharp shadows. Liana Lorigo extended this work by assuming the bottom of each image represented floor, and searched for areas with different colors or texture than the floor. However, an object near the robot could confuse it. Iwan Ulrich and Illah Nourbakhsh took the floor to be part of a previous image that had since been traversed. The work reported here is a first step towards automatically repeating and generalizing this thread.

1.1. Previous Work

Evolutionary Robotics

Evolutionary Robotics is a new field that uses simulated evolution to produce control programs for robots. Recent work can be found in [7] and [20]. Unless otherwise mentioned, the work reported here has been validated by running it on a real robot, not just in simulation. Most work uses bitstring Genetic Algorithms to evolve recurrent neural nets for obstacle avoidance and wall following using sonar, proximity or light sensors, e.g. [8, 17, 16]. Significantly, recurrent neural networks are considered much harder to train than feed forward networks, since gradient information typically isn t available. Evolutionary Computation doesn t use gradient information, and therefore even exploratory, toy problems use recurrence.

Nordin et al. [21] use a Genetic Programming variant that directly manipulates SPARC machine language. They use symbolic regression to predict the goodness of a state 300 ms in the future, based on the current sensor readings and action. For obstacle avoidance, the goodness is simply the sum of the proximity sensors, plus a term to reward moving quickly in a straight line. To choose a direction, the robot runs the best individual for all possible actions with the current sensor readings. The action with the highest estimated goodness is chosen. They use a population size of 10,000 and find that, in runs where perfect behaviour developed, it developed by generation 50.

Most work evolves in simulation, with the best individuals then run on a robot in the real world. Reynolds [24] has pointed out that without adding noise to a simulation, EC will find brittle solutions that wouldn t work on a real robot. Jakobi et al. [8] discovered that if there is significantly more noise in the simulation than on the real robot,

new random strategies become feasible that also don t work in practice.

As far as I know, no one has explored the reverse, i.e. whether entire classes of solution won t be found because they don t work in simulation. For example, dead reckoning error may be different on carpet then on a hard floor, so one possibility is to try to distinguish between them based on, say, their visual appearance. If this difference in error isn t modeled in the simulation, such a solution will never be found. Once evolutionary robotics gets beyond a basic stage, programs evolved in simulation may miss many subtle solutions.

As well, to my knowledge, no one has tried to simulate CCD camera images, either using standard computer graphics techniques or morphing previously captured images. The Sussex gantry robot [2] uses a CCD camera, but the images are reduced to the average brightness over three circles. These are significantly easier to simulate than a full CCD image, especially when the only objects are pure white on a black background. Smith [25] simulated a 16 pixel one-dimensional camera with auto iris on a robot soccer field. The 16 pixels were actually derived from 64 pixels; Smith doesn t say how. This is an important step, but again much easier than simulating a CCD image at, say, 160 x 120 pixels or above.

A few research groups perform all fitness evaluations on the real robot. Floreano and Mondada [4] evolve recurrent neural networks for obstacle avoidance and navigation from infrared proximity sensors. It takes them 39 minutes per generation of 80 individuals, and after about 50 generations the best individuals are near optimal, move extremely smoothly, and never bump into walls or corners. Naito et al. [19] evolve the configuration of eight logic elements, downloading each to the robot and testing it in the real world. Finally, the Sussex gantry robot [2] mentioned earlier has used evaluation on the real robot. They used a population size of 30, and found good solutions after 10 generations.

The closest work to that reported here was done by Baluja [1], who evolves a neural controller that interprets a 15 x 16 pixel image from a camera mounted on a car. The network outputs are interpreted as a steering direction, the goal being to keep it on the road. Training data comes from recording human drivers.

In summary, Evolutionary Robotics has used low bandwidth sensors, such as sonar or proximity sensors, presumably to cut down the amount of information to process. There are typically less than two dozen such sensors on a robot, and each returns at most a few readings a second. However, much traditional work in computer perception and robotics uses video or scanning laser range finders, which typically have tens to hundreds of thousands of pixels, and are processed at rates up to 10 Hz or more. Evolutionary Robotics has much to gain by scaling to these data rich inputs.

In addition, most Evolutionary Robotics has designed algorithms for simplified environments that are relatively easy to simulate. While evaluating evolved programs on real robots is considered essential in the field, those environments are typically still tailored for the robot. The current work attempts to evolve algorithms to interpret video of an unmodified office environment in real time, to help a robot wander while avoiding obstacles.

Visual Obstacle Avoidance

Somewhat surprisingly, there have only been a handful of complete systems that attempt obstacle avoidance using only vision in environments that weren t created for the robot. Larry Matthies group has built a number of complete systems, all using stereo vision [14]. They first rectify the images, then compute image pyramids, followed by computing sum of squared differences , filtering out bad matches using the left-right-line-of-sight consistency check, then low pass filter and blob filter the disparity map.

Their algorithm has been tested on both a prototype Mars rover and a HMMWV. The Mars rover accomplished a 100m autonomous run in sandy terrain interspersed with bushes and mounds of dirt [13]. The HMMWV has also accomplished runs of over 2 km without need for intervention, in natural off-road areas at Fort Hood in Texas [15]. The low pass and blob filtering mean the system can only detect large obstacles; a sapling in winter, for example, might go unseen.

Ratler [10] used a stereo vision algorithm to do autonomous navigation in planetary analog terrain. After rectification, the normalized correlation is used to find the stereo match. The match is rejected if the correlation or the standard deviation is too low, or if the second best correlation is close to the best. Travelling at 50 cm/sec over 6.7km the system had 16 failures, for a mean distance between failures of 417m. No information on failure modes is available.

David Coombs group at NIST has succeeded with runs of up to 20 minutes without collision in an office environment [3]. Their system uses optical flow from both a narrow and a wide angle camera to calculate time-to-impact, and provide feedback that rotationally stabilizes the cameras. Reasons for failure include the delay between perception and action, textureless surfaces, and hitting objects while turning (even while turning in place).

Liana Lorigo s algorithm [11, 12] assumes the bottom of the image represents clear ground, and searches up the image for the first window that has a different histogram than the bottom. This is done independently for each column. If the ground is mostly flat, then the further up the

image an object is, the further away it is. The robot heads to the side (left or right) where the objects are higher up.

Failure modes include objects outside the camera s field of view, especially when turning. Other failure modes are carpets with broad patterns, boundaries between patterns, sharp shadows, and specularities on shiny floors.

Ian Horswill s algorithm [5, 6] is similar to the above. It assumes that the floor is textureless, and labels any area whose texture is below threshold as floor. Then, moving from the bottom of the image up, it finds the first non-floor area in each column, turning left or right depending on which side has the most floor.

The system s major failure mode is braking for shafts of sunlight. In addition, it cannot break for objects it has seen previously but doesn t see now. Textureless objects with the same brightness as the floor also cause problems, as does poor illumination.

Ulrich and Nourbakhsh [27] took the floor to be the part of a previous image that had since been traversed.

Illah Nourbakhsh has used depth from focus for robot obstacle avoidance [22]. Three cameras, focused at different distances (near, middle and far), image the same scene. Whichever image is sharpest is the most in focus, so the objects are roughly at that distance. Actually, the images are divided into 40 windows (8 across and 5 down), which are treated independently, giving an 8 by 5 depth map.

In hundreds of hours of tests, the robot has avoided stair cases as well as students, often running down its batteries before a collision. However, failure modes include areas of low texture and tables at the robot s head height.

2. Experiments

All experiments were performed on the Uranus mobile robot, in the Mobile Robot Lab at Carnegie Mellon University. The robot has a three degree of freedom base with dead reckoned positioning. While forward/backwards motion and turning in place are fairly accurate (~ 1% error), sideways motion isn t (about 10-20% error, significant rotation). For sensing it uses a b/w analog video camera and a ring of 24 sonar sensors. Processing was done by an off board 700 MHz Pentium III computer running BeOS.

The work to date has taken place in a hallway whose most problematic features are glossy, textureless grey walls which often confound local depth estimation techniques such as stereo, optical flow and depth from focus.

2.1. The Evaluation of Learned Programs

One possible method of evaluating a learned program is to run it on a simulated robot in a virtual environment. The simulation could proceed faster than real time and doesn't require the constant supervision and resetting by hand that experiments with real robots often do. Many experiments with sonar and proximity sensors proceed in this way.

However, simulating CCD camera images to the fidelity required here is difficult, to say the least. Also, the creation of a simulation with noise levels and characteristics similar to those found in the real world is a time consuming and difficult task. Finally, as mentioned in the Previous Work section, certain subtle solutions may be possible in the real world, but not in simulation. Evolving in simulation may make the problem harder than need be, or just different.

In other words, simulations are necessarily different from the real world. The best evolved programs may not work in the real world, and the best program for the real world may do poorly in simulation. The task of constructing and refining a simulation to minimize these problems could prove interesting and valuable, but was not the approach chosen here.

Another option is to evaluate the algorithms by running them on the real robot. The robot could use the algorithm in question to navigate, stop when it hit something, and then travel back to where it started. When heading back, it could cheat by using a map of the space or additional sensors such as its sonar ring. Even so, this is very slow and requires a person to supervise the robot. The research groups that have attempted this use population sizes of less than 100 individuals, whereas GP typically uses sizes of 2000 to 10,000. Each evaluation on the robot takes a good fraction of a minute, and is prone to getting stuck, can t be done while its batteries are charging, can t be done at night, etc. As a rule of thumb, evaluations should take a second or less on average. This may actually be practical using a number of small, fast, reliable robots, but not with our beloved, lumbering Uranus.

For these reasons, the evolution is done off-line. Before the simulated evolution, the robot is run and vision and dead reckoning data is collected. During evolution, each evolved program is evaluated by executing it with the collected data as input, and comparing its output to a hand constructed correct answer. The output is interpreted as a distance to the nearest object, rather than a steering command directly, since this representation is more closely related to the input. The input, a set of greyscale values, may be so distantly related to the steering commands that the mapping is impossible to learn with current techniques and resources. With the success reported here, predicting the steering direction directly is an exciting next step.

2.2. System Description

The considerations of the previous section lead to the following setup. The robot is run autonomously with a

Table 1: Functions and Terminals of the Window Iteration Branch

root	`iterate-horizontal`, `iterate-vertical`
rectangle sizes	`r22`, `r23`, `r32`, `r33`, `r24`, `r42`, `r44`, `r55`, `r26`, `r62`, `r36`, `r63`, `r66`, `r77`, `r28`, `r82`, `r38`, `r83`, `r88`, `r2020`
arithmetic	`*`, `+`, `%`, `-`, `sqr` and random constants
parameters	`x-obstacle`, the horizontal pixel location in which to find the obstacle; `area`, the area of the window in pixels; `image-max-x` (319): `image-max-y` (239); `first-rect`, one if this is the first rectangle of the iteration, zero otherwise; `x` and `y`, the center of the rectangle in pixels.
flow control	`prog2`; `prog3`; `break`, with halts the execution of the branch, returning immediately without any more iterations; `if-le`
memory	`set-a` ... `set-e`, `read-a` ... `read-e`
image statistics	average and average-of-squared over the window: raw, truncated median, median corner, Sobel magnitude, and four directional Moravec interest operators.

hand-coded algorithm that uses sonar to avoid obstacles. During this online data collection run, camera images are continuously recorded. Then, during the offline learning run, simulated evolution evolves programs to estimate obstacle distance from each image. Finally, the best evolved programs are used to control the robot during online obstacle avoidance.

To collect images that are representative of what the cameras might see during that final stage, the robot collects data while avoiding obstacles under sonar. While obstacle avoidance under sonar is considered easier than under vision, it still took many attempts to get a working system. The method that proved most successful determines speed based on proximity to the nearest object, and determined direction of travel by fitting lines to points on the left and right sides of the robot. More details can be obtained from the author.

Each evolved program takes as input the image and the horizontal position of the column it must estimate. It returns a single number, the vertical position, in pixels, of the first (i.e. lowest) non-ground pixel. It s run on six different columns per image, on each of 75 images in the training set, for a total of 450 fitness cases. The fitness is the sum of the absolute differences between the returned values and ground truth. The absolute differences were capped at twenty.

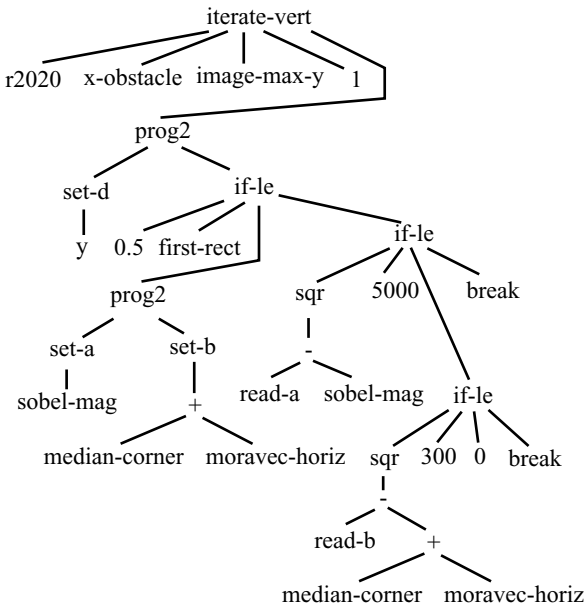

Figure 1: An example window iteration branch, used in a seed for the first generation of all runs.

The particular form of simulated evolution used here is Genetic Programming [9]. To give GP a little more structure, iterated window branches are included. These branches evaluate an evolved expression whose terminals include image statistics over a small window. The window is moved one pixel at a time, either horizontally or vertically, evaluating the expression at each location. They can read and write to five real valued registers, similar to Teller and Veloso s work with PADO [26]. The complete list of functions and terminals in the window iteration branch is shown in Table 1. An example iterated window branch, using a Lorigo style algorithm, is shown in Figure 1.

The return values of these expressions are discarded, so the register values are the only values left after execution. Each program has three window iteration branches, for a total of fifteen register values. These values are provided to the result producing branch, which must use them to estimate the location of the lowest non-ground pixel.

A Koza style tableau is shown in Table 2. The best evolved program is then run on the robot, and a hand written program converts the estimated object locations to speed and direction commands.

2.3. Experimental Setup

First, the camera was calibrated using the system described in [18]. The camera was then mounted on the robot, pitched 31 degrees down from horizontal. 75 images were collected, each 320 by 240 pixels, during a

Table 2: Koza Style Tableau

Objective	Given an image and a horizontal position within it, return the first non-ground pixel.
Architecture of individuals	Three *window iteration* branches and one result producing branch.
Terminal set for result producing branch	random constants, `x-obstacle`, `image-max-x` (319), `image-max-y` (239), `mem0a` ... `mem0e`, `mem1a` ... `mem1e`, `mem2a` ... `mem2e`
Function set for result producing branch	+, -, *, %, sqr, if-le
Fitness cases	Six columns in each of 75 images. 450 total.
Raw fitness	The sum, over the fitness cases, of the absolute value of the difference between the pixel location estimated by the evolved program and the hand created ground truth. If a difference was greater than 20, it was replaced by 20.
Standardized fitness	Same as raw fitness
Hits	The number of fitness cases for which the absolute difference was less than 2.0.
Parameters	101 generations, population size of 2000, tournament selection size 7, ramped-half-and-half with min size 6 and max size 9.

7.5 m data collection run. These images became the training set. The run was through a hallway and included two turns. Ground truth was then assigned by hand using a simple GUI. Typical images and ground truth are shown in Figure 2.

During offline learning, the images were first rectified to conform to an ideal perspective projection, and cropped to a horizontal field of view of 83 degrees using the above calibration information. The results of the operators were precomputed at every pixel, then the genetic programming run was started.

Offline learning was performed on a dual 700 MHz Pentium III, and evaluation times varied widely, but averaged approximately 725 msec. The time for a simulated evolution run also varied widely, averaging approximately 20 hours.

Figure 2: Training data, with ground truth indicated.

All genetic programming runs were seeded with the individual from Figure 1. That is, one of the 2000 individuals in generation zero has the window iteration branch show there, plus a result producing branch that simply returns the d register of that branch. This individual was designed as an example for explaining the approach and not intended to be run. The set of operators and their combination was chosen arbitrarily for their explanatory power without any thought as to how well they would work in practice. It was later used to test the system, and the thresholds were determined interactively at that time. The remaining individuals were created randomly, using the ramped-half-and-half method from [9].

After the offline learning, a hand written navigation algorithm used the estimates to decide speed and direction to travel. The best evolved algorithm was run on twelve columns in the image, twice as many as used in training. To better navigate around nearby obstacles, the camera was tilted further down, to 39.5 degrees from horizontal.

The navigation algorithm classifies estimates as either near (requiring an immediate halt), medium (slow to 2/3 speed to avoid collision) or far (avoid them before they become a problem.) This case based approach is inspired by the Property Mapping approach of Nourbakhsh [23]. If any of the middle four estimates are in the lower fifth of the image, or either of the two readings outside are at the bottom, then the object is considered near and the robot immediately halts. Otherwise, it looks for objects within four feet to the left and the right of where the robot is and where it would be if it continued straight. To convert pixel height in the image to real world distances, it assumes that the floor is flat, and that the non-ground object touches the floor. If an object is sighted, on either the left or the right, a line is drawn through the readings on each side, and the robot turns to run parallel to the lines.

If there are no objects near or in medium-sides, the algorithm looks for objects straight ahead within the far

boundary. Objects there cause it to respond by turning left or right, towards the largest gap. If all areas are clear other than far-sides, a line is fit to the side with the closest readings, and if the line is converging with the robots center line, the robot turns to move parallel. Finally, if there are no objects anywhere within the robot s field of view, it simply moves straight.

This algorithm was created by hand using traditional iterative design, and is still far from optimal. It is a natural application for simulated evolution, which is likely to do significantly better.

2.4. Results and Discussion

The seed individual performed better than any other individual in generation 0, although not by much. Of the 450 fitness cases (6 columns in each of 75 images), only two of them were within two pixels of the ground truth, the criterion for a hit. Its fitness was 14 pixels absolute error on average, where individual errors were limited to 20 pixels. By design, it fails in the many cases where the ground wasn t visible at the bottom of the column.

Twenty runs were completed. In all but one of these, the best-of-generation individual on the last generation did better than the seed. The average fitness (lower is better) of these twenty individuals was 7.43 pixels absolute error, the average number of hits (higher is better) was 111 out of a possible 450.

Interestingly, the three best best-of-run individuals had the same result producing branch as the seed, and iterated vertically in the desired column from bottom to top, just like the seed. This means the result producing branches only used one register from one of the three window iteration branches. Therefore, these best-of-run individuals have essentially the structure of Ian Horswill s and Liana Lorigo s systems.

The best individual from all runs had a standardized fitness of only 2.42 pixels absolute error per column, got 272 hits (i.e. 60.4% of estimates within 2 pixels), and had 587 nodes. After simplification (which didn t change the estimates it computes), 228 nodes remained. The rectangle size had been reduced from 20 by 20 pixels to 8 by 8. The first rectangle branch was considerably expanded, and after the first rectangle it only looked at the Sobel magnitude, i.e. the gradient.

The solution generalizes surprisingly well on the same camera in the same hallway. While its sensitive to the height of the camera, it is relatively insensitive to the pitch and the horizontal location of the column in the image. With the camera set to automatic gain it also provides acceptable results over a wide range of iris settings. It detects objects that weren t present during training, such as chairs or people, with about as much fidelity as it detects walls. It s also fast. Even without precomputing the image operators, the individual runs at about 10 Hz on a 333 MHz Pentium II. However, it s relatively sensitive to centimeter long pieces of metal or other small, shiny objects on the floor that produce high gradients.

With the camera fixed in one place, the algorithm produces occasional glitches, most often declaring that a pixel at the bottom of the image is non-ground when it is, in fact, ground. To stop these from causing too many panic halts, the hand written navigation algorithm filters readings by taking the minimum (highest pixel location) of consecutive estimates.

When navigating under sonar, fourteen sonar sensors for a total field of view of approximately 215 degrees, seeing well to the sides. While pitching the camera down increases awareness to the front left and front right of the robot, the area of awareness is still much smaller than with sonar, and entirely in front of the robot s base. In the reactive framework described here this makes it almost impossible to successfully navigate doorways, especially since the robot is only a few inches narrower than them. However, it performed very well at corridor following and avoiding obstacles such as people and chairs. The next revision of the system will include state, in order to ease these problems.

3. Bibliography

[1] S. Baluja, Evolution of an Artificial Neural Network Based Autonomous Land Vehicle Controller. IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics. 26, 3, 450-463. (1996)

[2] D. Cliff, P. Husbands and I. Harvey. Evolving Visually Guided Robots. In Proceedings of SAB92, the Second International Conference on the Simulation of Adaptive Behaviour. MIT Press, 1993.

[3] D. Coombs, M. Herman, T. Hong and M. Nashman Real-time Obstacle Avoidance using Central Flow Divergence and Peripheral Flow. Fifth International Conference on Computer Vision June 1995, pp. 276-83.

[4] D. Floreano and F. Mondada. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3. 1994.

[5] I. Horswill, Specialization of Perceptual Processes. Ph.D. Thesis, Massachusetts Institute of Technology, May 1993.

[6] I. Horswill, Polly: A Vision-Based Artificial Agent. Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), July 11-15, 1993.

[7] P. Husbands and J.-A. Meyer (Eds.), Evolutionary Robotics, Proceedings, First Euopean Workshop, EvoRobot98, Paris, France, April 1998.

[8] N. Jakobi, P. Husbands and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life, 1995.

[9] J. Koza, Genetic Programming, MIT Press, Cambridge, MA. 1992.

[10] E. Krotkov, M. Hebert & R. Simmons, Stereo perception and dead reckoning for a prototype lunar rover. Autonomous Robots 2(4) Dec 1995, pp. 313-331

[11] L.M. Lorigo Visually-guided obstacle avoidance in unstructured environments. MIT AI Laboratory Masters Thesis. February 1996.

[12] L.M. Lorigo, R.A. Brooks, and W.E.L. Grimson Visually-guided obstacle avoidance in unstructured environments. IEEE Conference on Intelligent Robots and Systems September 1997.

[13] L.H. Matthies, Stereo vision for planetary rovers: stochastic modeling to near real-time implementation. International Journal of Computer Vision, 8(1): 71-91, July 1992.

[14] L.H. Matthies, A. Kelly, & T. Litwin Obstacle Detection for Unmanned Ground Vehicles: A Progress Report. 1995.

[15] L.H. Matthies, Personal communication.

[16] L.A. Meeden, An Incremental Approach to Developing Intelligent Neural Network Controllers for Robots. IEEE Transactions of Systems, Man and Cybernetics Part B: Cybernetics. 26, 3, 474-485. (1996)

[17] O. Miglino, H. H. Lund and S. Nolfi, Evolving Mobile Robots in Simulated and Real Environments. Artificial Life , 2, 417-434 (1995).

[18] H.P. Moravec, DARPA MARS program research progress, http://www.frc.ri.cmu.edu/~hpm/project.archive/robot.papers/2000/ARPA.MARS.reports.00/Report.0001.html, Januray 2000.

[19] T. Naito, R. Odagiri, Y. Matsunaga, M. Tanifuji and K. Murase, Genetic Evolution of a Logic Circuit Which Controls an Autonomous Mobile Robot. Evolvable Systems: From Biology to Hardware. 1997.

[20] S. Nolfi and D. Floreano, Evolutionary Robotics. MIT Press / Bradford Books. 2000.

[21] P. Nordin, W. Banzhaf and M. Brameier, Evolution of a World Model for a Miniature Robot using Genetic Programming. Robotics and Autonomous Systems, 25, pp. 105-116. 1998.

[22] I. Nourbakhsh, A Sighted Robot: Can we ever build a robot that really doesn t hit (or fall into) obstacles? The Robotics Practitioner, Spring 1996, pp. 11-14.

[23] I. Nourbakhsh, Property Mapping: A simple technique for mobile robot programming. In Proceedings of AAAI 2000.

[24] C. W. Reynolds, Evolution of Obstacle Avoidance Behavior: Using Nosie to Promote Robust Solutions. In Advances in Genetic Programming, MIT Press, pp. 221-242. 1994.

[25] T. M. C. Smith, Blurred Vision: Simulation-Reality Transfer of a Visually Guided Robot. In Evolutionary Robotics, Proceedings of the First Euopean Workshop, EvoRobot98, Paris, France, April 1998.

[26] A. Teller and M. Veloso, PADO: A new learning architecture for object recognition. In Symbolic Visual Learning, Oxford Press, pp. 77-112. 1997.

[27] I. Ulrich and I. Nourbakhsh, Appearance-Based Obstacle Detection with Monocular Color Vision. In Proceedings of AAAI 2000.