Available online at www.sciencedirect.com

SCIENCE DIRECT®

Pattern Recognition Letters 27 (2006) 1174–1180

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Evolving visual sonar: Depth from monocular images

Martin C. Martin *

Mobile Robotics Laboratory, Carnegie Mellon University, Pittsburgh, PA, United States

Available online 27 December 2005

Abstract

To recover depth from images, the human visual system uses many monocular depth cues, which vision research has only begun to explore. Because a given image can have many possible interpretations, constraints are needed to eliminate ambiguity, and the most powerful constraints are domain specific. As an experiment in the automatic discovery and exploitation of constraints, genetic programming was used to find algorithms for obstacle detection. The algorithms are designed to be a replacement for sonar, returning the location of the nearest obstacle in a given direction.

The evolved algorithms worked surprisingly well. Errors were largely transient. The algorithms generalized to both novel views of the office environment and to unseen obstacles. They were combined with a simple reactive wandering program originally written for sonar. The result exhibited good performance in an office environment, colliding only with obstacles outside the robot's field of view.

Time to collision results and failure modes are presented. Code is available for download.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Genetic programming; Robotics; Visual navigation; Monocular vision

1. Introduction

Video cameras are attractive sensors that provide a large amount of data at high speed and low cost. These properties make them particularly attractive in robotics. However, current robotics navigation algorithms, from potential fields (Arkin, 1999) to simultaneous localization and mapping (SLAM) (Thrun, 2002), need the distance to the first obstacle in various directions.

To extract distances, stereo vision is typically used. Stereo vision returns a dense depth map, but requires precise synchronization and calibration to find the corresponding line to within a few pixels. It also requires significant CPU power. Yet a full depth map is not needed, as sonar sensors allow reliable navigation. Can we produce a kind of "visual sonar", a simple algorithm that produces a few depth estimates per image?

Monocular images clearly contain sufficient information, as people can understand them and psychologists have elucidated many monocular depth cues. Yet it is far from obvious how to compute depth from such an image. Monocular images were central to early work in computer vision, where the problem was commonly recognized as under constrained, in that many different physical scenes can result in the same image. For example, two scenes that differ only by having one object closer to the camera and smaller can produce exactly the same image.

The solution was to incorporate *constraints*, such as that chimneys sit on top of houses, rather than hover in the air in front of them. Marr considered constraints "the critical act in formulating computational theories" of vision (Marr and Nishihara, 1978). And the second half of the influential book Duda and Hart (1973) focused on finding the constraints necessary for scene understanding. Yet today computer vision researchers look for domain independent techniques since these have the most general applicability.

Previous work in monocular vision (summarized below) relies on the researcher to discover domain constraints and embody them in algorithms. To create algorithms for new

* Present address: Icosystem Corporation, 10 Fawcett Street, Cambridge, MA 02138, United States.
E-mail address: martin@metahuman.org
URL: www.metahuman.org/martin.

0167-8655/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2005.07.015

environments requires someone with a background in Robotics. This severely restricts the practical application of the algorithms. This paper provides a way out of the tension between domain dependence and independence, by proposing a general, domain independent tool which searches a set of algorithms to discover which constraints to exploit.

The main contribution of this work is the *automatic* discovery of domain constraints. This paper uses genetic programming to create obstacle detection algorithms. In particular, we created a function that takes in a 320×240 grayscale image, as well as the horizontal location of a column in the image, and returns the lowest non-ground pixel in the image. Thus, we define an obstacle as anything other than the ground, be it a chair, a wall or a person. See Fig. 1. We used the simplest, most common form of genetic programming, described in (Koza, 1992). The only significant addition was a node for iterating a window over an image. Moving a window over an image is a common component of many vision algorithms. This work extends earlier efforts (Martin, 2002) by extending the previous analysis to explore issues of interest in pattern recognition.

The remainder of this section reviews previous work in both monocular vision and the evolution of visual routines. Section 2 describes the details of the evolutionary computation setup. Section 3 describes the performance of the system on the training data, as well as the results of using the evolved vision algorithm to control a robot. Finally, Section 4 analyzes the results and discusses their implications.

1.1. Previous work

A few research groups have tackled monocular vision for robotics. Polly the robot (Horswill, 1993) looked for the lowest obstacle in a given column of an image. This exploits a documented monocular cue: if most obstacles touch the floor and the floor is roughly flat, then obstacles higher in the image are further away (Ooi et al., 2001).

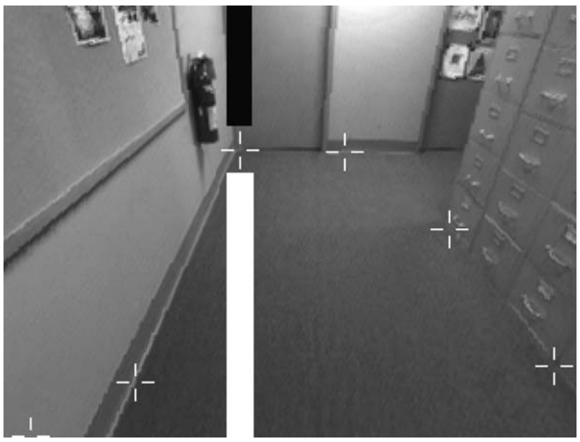
Fig. 1. Evolved programs return the pixel coordinates of the lowest obstacle in a given column of the image. In the above image, the white bar covers ground pixels, while the black bar covers obstacles. The cross between these two shows the exact transition point. The five other crosses show the ideal response for five other columns.

To find the lowest obstacle, Polly's creator assumed that the carpet was always textureless, and thus declared any area with visual texture to be an obstacle. Although algorithmically quite simple, it allowed Polly to give over 100 tours of the seventh floor the MIT Artificial Intelligence Lab, logging hundreds of hours without incident (Horswill, 1994). Lorigo (1996) assumed the bottom of the image was always ground, and looked for areas that differed significantly, declaring them obstacles. Ulrich and Nourbakhsh (2000) took ground to be the portion of a previous image that the robot has already traversed.

The work in (Xie, 1996) used a stereo pair of cameras, but without matching windows between the two images. Instead, it projected features from one image into the other, at the location expected if the feature were on the ground plane. When features fail to match up, this indicates they are not on the ground plane, suggesting that they are an obstacle. While this reduces the computational burden, it still requires careful calibration of the cameras and their positions relative to the ground plane. If either the cameras or the objects are moving, it also requires temporal synchronization of the cameras.

The beebot (Camus et al., 1999) used the divergence of the optical flow field to compute time to impact, steering where this was lowest. It has wandered the lab at 20 cm/s for as long as 26 min without collision. Sobey (1994) used a monocular camera combined with a zigzag motion to estimate the range to objects, and estimate the safe distance of travel in the present direction.

Others have evolved visual routines before. Johnson et al. (1994) evolved programs to find hands in binary images of people. The images were generated by having people stand in front of a blue screen, and marking every non-blue pixel as part of the person. Nodes included the centroid and bounding box of the silhouette, horizontal edge detectors and the ability to find the leftmost, rightmost or average of a set of detected edges. Their results were promising, correctly classifying up to 93% of images. That performance was better than the best algorithms they were able to write by hand.

Cliff et al. (1993) evolved programs that allowed the Sussex gantry robot to move toward a white square while avoiding a white triangle in an otherwise black world. Images from the CCD camera were first reduced to three numbers, the average brightness over three circles. Smith (1998) used a 16 pixel one-dimensional camera on a robot soccer field. Several people have evolved convolution masks, such as interest operators and edge detectors, that return a result for every pixel (Bala et al., 1996, 1997; Ebner, 1998; Harris and Buxton, 1996; Harris, 1997; Benson, 2000; Guarda et al., 1998). Teller and Veloso (1997) evolved programs to classify images which contained functions to return the min, max, mean and variance of image pixels over a rectangle. They also made use of an array of memory, as a way of maintaining state.

2. Methods

A supervised learning framework was employed. Images of three different environments were obtained, and the location of the lowest non-ground pixel determined manually. Evolutionary computation was then given these images, and the output compared to these manually determined responses. The best evolved programs were then used to control a robot, whose performance was measured.

Images were recorded with a black and white analog video camera. Processing was performed by an off board dual 700 MHz Pentium III computer. Evolved algorithms took in an image, that is, a 320×240 array of grayscale values, and the horizontal location within an image of a column. They returned an estimate of the lowest non-ground pixel in that column of the image, see Fig. 1.

The camera was mounted on the robot and pitched 51° down from horizontal. The training set was gathered while the robot navigated using sonar, not vision. The view from the camera, a series of 320×240 grayscale images, was passively recorded. A detailed description of the navigation algorithm can be found in (Martin, 2001).

Three datasets were used. The first two consisted of images from two hallways in different buildings, Newell Simon Hall (NSH) and the Field & Mobile Robotics Building (FMRB). The third data set consisted of every other image from the first two. Both environments sported glossy, textureless gray walls, which are notoriously difficult for local depth estimation techniques such as stereo and optical flow. The robot made two or three turns in each environment, and its shadow is clearly visible in most frames. In NSH there were several people visible at different times. In that same environment, the carpet had a large red stripe at one end of the hallway. In FMRB, the overhead lights were burned out in one part of the hallway, so that the intensity of the carpet varied widely.

To establish ground truth, the author determined and recorded the ideal response of the vision subsystem, i.e., the location of the lowest non-ground pixel, in each of six columns in each image. The number six was chosen somewhat arbitrarily to match the typical number of sonar sensors in the camera's roughly 90° field of view. The three training sets had approximately 70 images each. The total number of fitness cases was therefore approximately $70 \times 6 = 420$ per data set.

Each evolved program was run six times on each image, once for each column where training data was assigned. Fitness was the number of "correct" responses. A column was judged correct if the value returned was within 10 pixels of the manually determined ideal response.

The vision algorithm was evolved using genetic programming (Koza, 1992). Genetic programming is a genetic algorithm in which individuals are programs, represented as parse trees. The set of node types is chosen by the programmer and is dependent on the problem domain. Random programs are created by choosing a random node type for the root, then choosing a random node type for each of its arguments, continuing recursively. A fixed number (say 4000) are created this way and each one is assigned a scalar fitness. Individuals are then randomly chosen, with the fitter individuals more likely to be chosen, and are either copied verbatim into the next population (replication) or a random subtree is chosen in two different individuals and swapped (crossover). Once 4000 individuals are created this way, the original individuals are discarded, the new individuals are assigned fitness and the process repeats.

A new type of node, dubbed the *iterate* node, was created, which moved a rectangle vertically over the image. The node's three children were the size of the rectangle, the horizontal location (in pixels) of the column in which to iterate, and a subtree to execute at each location. This last argument could incorporate the results of various image operators applied to the window.

The full list of node types (i.e., function and terminal sets) is summarized in Table 1. In addition to the iterate

Table 1
The set of node types that genetic programming combined into programs

Root	`iterate-up`, `iterate-down`
Rectangle sizes	2×2, 2×3, 3×2, 3×3, 2×4, 4×2, 4×4, 5×5, 2×6, 6×2, 3×6, 6×3,6×6, 7×7, 2×8, 8×2, 3×8, 8×3, 8×8
Arithmetic	+, −, ×, % (protected division), `square`, `sin`, `cos`, `tan`, `atan`, `atan2`, `min`, `max`
Parameters	`x-obstacle`, the horizontal pixel location in which to find the obstacle; `window-area`, the area of the window in pixels; `image-max-x` (always 319): `image-max-y` (always 239); `first-window`, +1 if this is the first location of the window for this iteration, −1 otherwise; `x` and `y`, the center of the rectangle in pixels; random constants
Flow control	`prog2`; `prog3`; `break`, which halts the execution of the tree, returning immediately without any more iterations; `if−le`, the if less than or equal to operator
Memory	`set-a` …`set-e`, `read-a` …`read-e`
Results of first tree	`iter2-a` …`iter2-e` (only available in the second tree)
Image operators	Average and average-of-squared over the window: raw, truncated median, median corner, Sobel magnitude, and four directional Moravec interest operators

Table 2
Summary of the evolutionary run

Objective	Given an image and a horizontal position within it, return the first non-ground pixel in that column
Architecture of individuals	Two trees, each with a single iterate node at the root
Fitness cases	Six columns in each of 65–71 images, 390–426 total
Raw fitness	The percentage of fitness cases correct. A fitness case was correct if the pixel location estimated by the evolved program was within 10 pixels of the author's determination of the ideal answer
Standardized fitness	100% minus raw fitness
Miscellaneous parameters	51 generations, population size of 4000, tournament selection with tournament size of 7, ramped-half-and-half with min size 6 and max size 9

node and rectangle sizes, the list includes standard arithmetic operators, various parameters of the image, flow control, nodes to set and read five floating point registers, and both the average, and average of squared value, of common image operators. The registers were inspired by Teller and Veloso (1997).

Each individual was composed of two trees. Each tree had a single iterate node, always at the root, and this rule was enforced during creation and crossover of individuals. During evaluation, the first tree was run, then the second tree, which could use the results of the first. Using two trees allows two pass algorithms, for example estimating properties of areas likely to be floor (such as the bottom middle of the image), then comparing these to the desired column. Iterate nodes do not return a value in the traditional sense. Instead, the final values of the five registers are used as five return values from the tree. Thus, the second tree can use the values of any of the five registers of the first tree. The result of the entire individual was simply the final value of the *a* register of the second tree.

The other parameters used during evolution are summarized in Table 2. The miscellaneous parameters are those used by Koza (1994), with the exception that we used a larger population. Early, informal experiments with a population size of 2000 showed poor performance, while the larger population size worked quite well. On a dual 700 MHz Pentium III the times to assign fitness to a single individual varied widely but averaged 725 ms. The time for an entire run of 51 generations also varied widely, averaging approximately 20 h.

3. Results

All three experiments achieved a fitness of greater than 85%. They did this despite burned out lights and other effects that caused the carpet's average intensity to vary from zero to 145 out of 255; despite large gradients caused by imaging artifacts; despite moir patterns of image noise; and despite the shadow of the robot.

On the FMRB data set, in all but two of the runs, the best individual in the initial population achieved a fitness of 21.6%. The other two runs achieved 29.3% and 36.4% in the initial population. The runs on the other two datasets were similar.

The best individual from all FMRB runs achieved 91.8%. Some examples from the training set are shown in Fig. 2. While hard to see, the white squares are the size of the window chosen by evolution. The other two conditions did almost as well, achieving 90.5% on the NSH data set, and 84.8% on the combined data set. In all three experiments the errors were transient, that is, an error in one image would disappear in the next image, even when the same objects were visible. As is discussed below, this means the errors are from noise in the imaging process and are easily filtered out. The exception of the stripe of red carpet in Newell Simon Hall, which was uniformly considered an obstacle when near the camera.

3.1. Validation on the robot

All three individuals generalized surprisingly well when run on new video of the same hallway. They were relatively insensitive to the pitch and the horizontal location of the column in the image. They worked on a range of iris settings and camera tilts and did not overfit to column location. They detected obstacles that were not present during training, such as chairs or people, with only a little less fidelity than they detected walls. They were also fast, running at about 10 Hz on a 700 MHz Pentium III.

If most obstacles touch the ground, then given rough estimates of the camera's height and tilt angle, these image locations can be translated into distances. A hand written, reactive algorithm used these distances to avoid obstacles in real time. The algorithm was originally developed using sonar, and ran with the distances from evolved programs with very little tweaking. A detailed description of the algorithm can be found in (Martin, 2001). The three different evolved programs, trained on the different data sets, all used the same avoidance algorithm. The vision subsystems were run on 12 columns in the image, twice as many as used in training, as six columns allowed rather large objects to pass between the columns.

The robot was then run in the same environment(s) it was trained in. Videos of this online validation can be found at www.metahuman.org/martin. These routes included views of parts of the environment never seen during training. Obstacles were present that were not present during training, such as chairs, trash cans and people. It

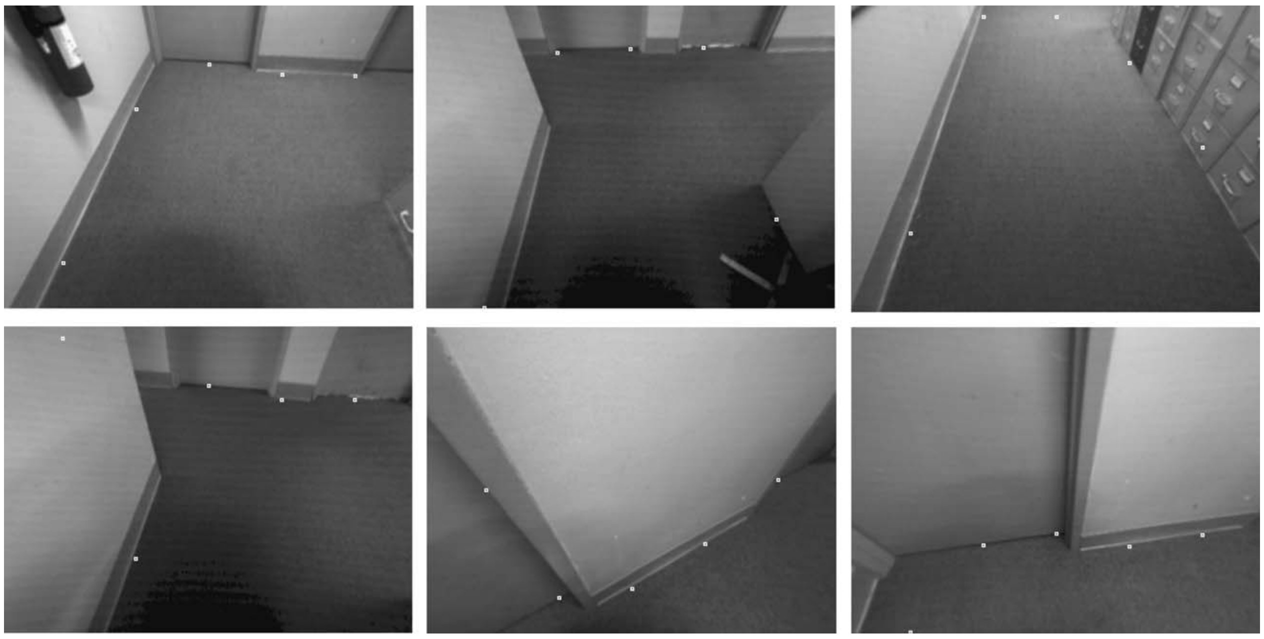
Fig. 2. Results of the best evolved individual on test (i.e., not training) data. The evolved program achieved the correct answer over 90% of the time.

avoided these obstacles well. Most errors were also conservative, i.e., declaring that a pixel was an obstacle when it was, in fact, ground.

Most errors were also transient, lasting only a frame or two even when the camera was stationary. As mentioned previously, the exception was the red carpet in the NSH data set, which was handled poorly even on the training data. It was classified as an obstacle when nearby, causing the navigation to consistently treat it as a wall. In five validation runs, the mean distance and time to collision for the FMRB individual was 133 m, 20:01 min, and the longest was 260 m, 39:32 min. These are comparable to the published performance of the hand-written systems.

Not surprisingly, the vision subsystem from the combined data set performed a little worse in each environment than the vision system developed from data only in that space. It did not have a new class of error, but instead was more likely to make one of the errors made by the other vision subsystems. In the FMRB hallway, its average distance and time to collision were 72 m and 10 min respectively.

3.2. Structure of best evolved individuals

The best individual from the FMRB condition was simplified. The resulting program was quite readable, see Fig. 3. The two trees used two very different techniques. The second tree, whose *a* register holds the obstacle location, was essentially a decision tree, although one of the decisions has a non-linear boundary in the space spanned by two image operators. The other tree, which detected obstacles at the bottom of the image, was a recurrent mathematical function involving both a single image operator and the location of the column.

The second tree uses the non-directional gradient magnitude as a primary indication of an object, although at intermediate values (between 239 and 414) is uses a non-linear comparison with a directional gradient. However, gradients are ignored when the median image brightness is less than 35. On closer inspection, this resolved a problem with the image digitizer which would round any pixel of brightness less than 32–16. This produced areas of brightness 16 next to pixels of brightness 33, creating artificial gradients. Ignoring areas darker than 35 eliminates these spurious gradients.

The other best-of-experiment individuals were similarly simplified. Although all individuals were composed of two trees, most individuals ignored the results of the first tree. Thus, one tree would most likely have been sufficient. Evolution had exploited a number of techniques. The best NSH individual was a sequence of if–then conditions similar to a decision tree but involving non-linear combinations of up to five different image operators. In all cases, the bottom row of the image was handled using different code than the rest of the image. The mechanisms for this varied; on the FMRB data set it used two different trees for the two conditions, whereas in the NSH and combined experiments a multitude of `if–le` statements were used to run different code at different locations. Interestingly, the raw image statistic was never used, although the median filtered statistic was. All directional gradients of the image intensity were used except the vertical.

4. Discussion

This work explored a simple methodology for automatically discovering algorithms that exploit domain constraints. Genetic programming was used to search

```
First tree:

$b$ (initial value) := $3.40282 \times 10^{38}$

Iterate-Up, 3×3 window, in column $desired\text{-}x$:
$b := \frac{b(1+h)-(1+h+h^2+h^4)desired\text{-}x}{h^5}$;

Where $h$ is $moravec\text{-}horizontal$.

Second tree:

Iterate-Down, 3×3 window, centered on the desired column:
if $y \leq 9$ then
    $a := y$;

if $iter2\text{-}b > 9$ then
    $a := y$;

if $median > 35.4444$ then
{
    if $gradient > 413.96$ then
        $a := y$;

    if $gradient > 239$ and $(gradient/239)^4 > moravec\text{-}slash$ then
        $a := y$;
}
```

Fig. 3. The best individual from all runs on the FMRB data set. This individual has been simplified without changing its behavior. The meanings of the various identifiers are listed in Table 1. The best individuals evolved on the other data sets had more complex structure, making greater use of memory.

through a space of monocular vision algorithms. In all three environments, evolution produced algorithms which achieved over 85% fitness. Even on a new stream of images, all errors lasted only a single frame except for the red carpet in the NSH environment. These algorithms were then combined with a reactive obstacle avoidance algorithm originally developed for sonar. The combined system worked surprisingly well, generalizing to areas of the environment not seen during training. Performance was comparable to that of existing hand coded systems from the literature. The code used for this work may be downloaded from www.metahuman.org/martin.

Evolution generated a number of different algorithm styles, from recurrent mathematical expressions to a decision tree with non-linear decision surfaces. In all cases, the bottom row of the image was handled using different code than the rest of the image. This division into two cases—boundaries between obstacles and floor, vs. obstacles that go to the bottom of the image—was discovered automatically. Nothing in the representation suggested the two different cases. Instead, this reflects a natural dichotomy in the images: at the very bottom of the image a program must detect the presence or absence of an obstacle, but in the middle of an image it must detect the *transition* between ground and non-ground, which is likely to have a significant gradient. This division was not noticed by any of the three groups mentioned in the introduction who created Polly style algorithms by hand, yet in retrospect it is obvious and natural.

This work is also the first example of the evolution of a complete vision system for a pre-existing environment. Earlier evolutionary work, such as that by Cliff et al. (1993) and Smith (1998) used simple environments constructed especially for the robot, and as a result, could process at most 16 pixels.

This work can also be seen as automating the construction of Polly style algorithms. The design of Polly's vision system required someone with knowledge of robotics and computer vision, whereas the current work only needs someone to mark obstacle location in an image with a simple GUI.

Genetic programming uses what amounts to a scripting language for a representation. This leads to very different results than general function approximation such as neural networks. For example, if a given input is largely unrelated to the output, genetic programming will often ignore the input altogether, whereas a neural network will simply fit to the noise. In searching the space of expressions, genetic programming is using a representation that mathematicians and computer scientists feel is natural.

The vision sensor is cheap, a simple monocular, 320×240 grayscale camera such as a Webcam. This contrasts sharply with stereo vision platforms such as the Triclops which cost thousands of dollars due to need for precision synchronization and calibration. In contrast, our calibration could be performed by any teenager with a protractor and yardstick.

This simplicity comes from exploiting domain knowledge. Computer vision is currently focused on domain independent methods, yet these have limits. This work provides a way out of the tension between domain dependence and independence, by proposing a general, domain independent tool which analyzes a domain to discover which constraints to exploit.

References

Arkin, R.C., 1999. Behavior-Based Robotics, second ed. The MIT Press, Cambridge, MA.

Bala, J.K., DeJong, K., Huang, J., Vafaie, H., Wechsler, H., 1996. Visual routine for eye detection using hybrid genetic architectures. In: Intl. Conf. on Pattern Recognition, Vienna, Austria.

Bala, J.K., DeJong, K., Huang, J., Vafaie, H., Wechsler, H., 1997. Using learning to facilitate the evolution of features for recognizing visual concepts. Evol. Comput. 4 (3), 297–312.

Benson, K., 2000. Evolving automatic target detection algorithms that logically combine decision spaces. In: Proc. 11th British Machine Vision Conf., Bristol, UK, September 2000, pp. 685–694.

Camus, T., Coombs, D., Herman, M., Hong, H.T., 1999. Real-time single-workstation obstacle avoidance using only wide-field flow divergence. J. Computer Vision Res. 1 (3).

Cliff, D., Husbands, P., Harvey, I., 1993. Evolving visually guided robots. In: Proc. SAB92, the Second Intl. Conf. on the Simulation of Adaptive Behaviour. MIT Press.

Duda, R.O., Hart, P.E., 1973. Pattern Classification and Scene Analysis. John Wiley & Sons.

Ebner, M., 1998. On the evolution of interest operators using genetic programming. In: Poli, R. et al. (Eds.), EuroGP'98: The First European Workshop on Genetic Programming, Paris, France, 14–15 April 1998, pp. 6–10.

Guarda, A., Le Gal, C., Lux, A., 1998. Evolving visual features and detectors. In: Intl. Symp. on Computer Graphics, Image Proc. and Vision, Rio de Janeiro.

Harris, C., 1997. An Investigation into the application of genetic programming techniques to signal analysis and feature detection. Ph.D. Dissertation, University College London, London, UK.

Harris, C., Buxton, B., 1996. Evolving edge detectors. Technical Report Research Note RN/96/3, University College London, London, UK, January 1996.

Horswill, I., 1993. Specialization of perceptual processes. Ph.D. Dissertation, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, May 1993.

Horswill, I., 1994. Visual collision avoidance by segmentation. In: IROS '94: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 902–909.

Johnson, M.P., Maes, P., Darrell, T., 1994. Evolving visual routines. In: Brooks, R., Maes, P. (Eds.), Artificial Life IV, Proc. Fourth Intl. Workshop on the Synth. and Simul. of Living Systems. Bradford Books.

Koza, J.R., 1992. Genetic Programming. MIT Press.

Koza, J.R., 1994. Genetic Programming II. MIT Press.

Lorigo, L., 1996. Visually-guided obstacle avoidance in unstructured environments. MIT Master's thesis, Artificial Intelligence Laboratory.

Marr, D., Nishihara, K., 1978. Visual information processing: Artificial intelligence and the sensorium of sight. Technol. Rev. 81 (1), 2–23.

Martin, M.C., 2001. The simulated evolution of robot perception. Ph.D. Dissertation, Carnegie Mellon University, Robotics Institute.

Martin, M.C., 2002. Genetic programming for robot vision. In: 7th Int'l. Conf. on the Simulation of Adaptive Behaviour, Edinburgh.

Ooi, T.L., Wu, B., He, Z.J., 2001. Distance determined by the angular declination below the horizon. Nature 414 (8), 197–200.

Smith, T.M.C., 1998. Blurred vision: Simulation-reality transfer of a visually guided robot. In: EvoRobot98: Proc. 1st European Workshop on Evolutionary Robotics, Paris, France, April 1998.

Sobey, P.J., 1994. Active Navigation with a Monocular Robot. Biol. Cybernet. 71 (5), 433–440.

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

Thrun, S., 2002. Robotic mapping: A survey. In: Lakemeyer, G., Nebel, B. (Eds.), Exploring Artificial Intelligence in the New Millennium. Morgan Kaufmann.

Ulrich, I., Nourbakhsh, I., 2000. Appearance-based obstacle detection with monocular color vision. In: Proc. AAAI National Conference on Artificial Intelligence, Austin, TX, July/August 2000.

Xie, M., 1996. Matching free stereo vision for detecting obstacles on a ground plane. Machine Vision Appl. 9 (1), 9–13.