Machine Vision: Exploring Context with Genetic Programming

Daniel Howard and Simon C. Roberts
Software Evolution Centre
Building U50, QinetiQ
Malvern, Worcs. WR14 3PS
United Kingdom
dhoward@qinetiq.com
Phone: +44-1684-894480

Conor Ryan
University of Limerick
Limerick
Ireland

Abstract

This paper proposes an advanced method of object detection using a *data crawler*. Starting from a preliminary 'object identification', the data crawler scrutinizes the object's surroundings, and places flags where its interest has been aroused. Next, the binary map defined by these flags is analyzed statistically to quantify the context in which the object appears. The crawler's overall output indicates whether the object is a required target or a false alarm. The crawler's navigator program, its flag-placement program and its target detection program, are all controlled by a tree-based Genetic Programming (GP) method with fixed architecture Automatically Defined Functions (ADFs).

1 Introduction

The perennial problem with machine vision is to distil features which characterize an object from a huge amount of available information, i.e. the specification of an intelligent data reduction mechanism. This data reduction must be discovered because it cannot be determined *a priori*.

Moreover, if an object is viewed too closely, there is not enough contextual information upon which to make an identification. Conversely, if an object is too far away, information overload makes identification equally difficult. To deal with this surfeit of information, a way of selecting a sub-set of data, which is representative of an object, needs to be derived which allows the object to be identified.

In the context of GP, (Tackett, 1993, Poli, 1996, Daida et al., 1996, Howard and Roberts, 1999) settled for very practical schemes which manipulated statistics computed from pixel data, and (Andre, 1994, Johnson et al., 1994, Teller and Veloso, 1996, Harris and Buxton, 1996) pursued other evolvable object detection themes. (Roberts and Howard, 2000) also exploited the evolutionary paradigm to obtain the orientation of poorly defined objects such as vehicles in IR imagery. (Benson, 2000) exploited the software reuse paradigm (Koza, 1994) by embedding tree GP inside an evolvable Finite State Machine, a scheme that parallels the Automatically Defined Functions (ADFs) idea. (Roberts et al., 2001) successfully exploited problem modularities and regularities for a difficult object detection task with the subtree encapsulation idea.

This paper draws on the ADF idea to arrive at a mechanism of exploring the surrounding context and a more informed object detection (Howard et al., 2002). As post-processor to the (Roberts and Howard, 1999) scheme, it aims to intelligently lower that scheme's false alarm rate. The (Roberts and Howard, 1999) scheme processes infrared line-scan (IRLS) imagery acquired by low-flying aircraft to detect land vehicles. The scheme produces many false alarms because its aim is to detect any region which represents some part of a vehicle. Users of this scheme know that objects may be obscured by other objects accidentally or deliberately, and so a high false alarm rate is acceptable. However, the scheme cannot combine clues together to eliminate obvious misidentifications, e.g. a vehicle cannot be parked on a roof top. So the proposition is that obvious false alarms may be eliminated by exploring the context of misidentifications.

2 Overview and inspiration

It is clearly impossible to identify the vehicle object when the image is viewed from too close, for example, as shown for the vehicle detection problem in Figure 1.

Animal vision probably attempts to identify a number

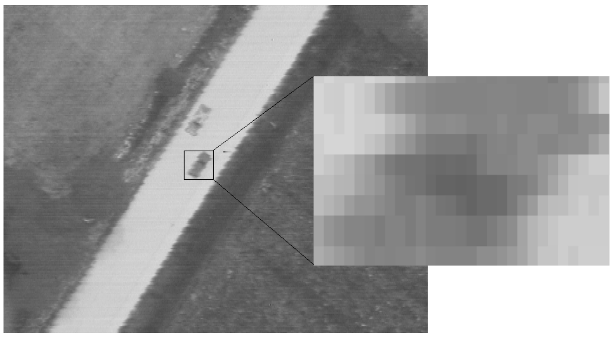

Figure 1: Two different views: close-up and from afar.

of very specific image clues that resemble some close-up detail of an object. The brain can then interpret these clues to 'imagine' the object from afar. In other words, object recognition involves constructing a mental model of an object, and superimposing this model onto the evidence perceived from the real-world. The scheme proposed in this paper is based on this idea of using both views, near and far, and of using a small number of image clues to connect these views to detect the object. The central problem then becomes how to identify the required image clues?

Evolution with GP tackles this difficult issue. Starting from the location of an object identification by the (Roberts and Howard, 1999) detector, a 'data crawler' inspects the image surroundings to discover a few useful clues. It marks a 2D binary map to flag the location of discovered 'clues', and constructs statistics based on their distribution. These statistics are also combined by GP to arrive at a decision concerning the originating pixel: whether to confirm or reject that the pixel belongs to a target object.

The evolutionary process is driven by the global objective of reducing misses and false alarms. It is implemented using GP with fixed architecture ADFs (Koza, 1994). The GP system decides the following. How should this image crawler be guided in its search for the clues? How far should it travel and turn? What should excite it sufficiently to cause it to flag the existence of an image clue?

3 Structure and representation

Each individual data crawler in the population is a GP tree structure, equipped with two result-producing branches and two ADF branches:

- First Result Branch (FRB)
- Turn Decision Branch (TDB)
- Mark Decision Branch (MDB)
- Second Result Branch (SRB)

Table 1: Function sets. GL represents glue functions.

Branch	Functions
FRB	$GL2$, $GL3$, MDB
TDB	+, -, *, / ($x/0 = 1$), min(A, B), max(A, B), if ($A < B$) then C else D
MDB	+, -, *, / ($x/0 = 1$), min(A, B), max(A, B), if ($A < B$) then C else D, WRITEMEM, READMEM
SRB	+, -, *, / ($x/0 = 1$), min(A, B), max(A, B), if ($A < B$) then C else D

FRB is allowed to call TDB and MDB many times but otherwise the branches are unrelated. SRB works on the 2D memory devised by FRB as will be explained shortly. Each branch determines a specific property of the crawler, and each has its own set of terminals and functions. Moreover, some have access to task specific working memories:

- FLAG memory
- WORKING memory
- MOVE memory

Each individual maintains these memories to save information about its past experience. Data crawler decisions, e.g. whether or not to mark the image with a flag, require a memory of past events which allows the data crawler to consult and integrate previous information prior to a decision.

FRB, working with the TDB and MDB, guides the data crawler to explore the image in the near field, and to deposit a number of 'flags' or image clues. Once this process is completed, the discriminant SRB looks from afar at this binary 2D map of 'flags' and 'no flags'. The SRB then decides whether the original starting point was correctly indicated as a true target, or whether it was a false alarm. Tables 1 and 2 give the function and terminal sets for each GP branch.

3.1 First Result Branch (FRB)

The data crawler has similarity with the Santa Fe trail ant in (Koza, 1992), e.g. the glue functions $GL2$ and

Table 2: Terminal sets (see text).

Branch	**Terminals**
FRB	M, TDB
TDB	μ_{11}^C, σ_{11}^C, μ_{11}^N, σ_{11}^N, μ_{11}^E, σ_{11}^E, μ_{11}^S, σ_{11}^S, μ_{11}^W, σ_{11}^W, READFLAG, READMOVE, altitude in feet (250:650)
MDB	μ_{disk}, σ_{disk}, altitude in feet (250:650), READFLAG, READMOVE
SRB	10 flag based terminals, altitude in feet (250:650), 10 textural area statistics

$GL3$ and the M or 'move terminal' to move the crawler one pixel in the direction of its travel. However, the L (left) and R (right) terminals used to turn the Santa Fe trail ant are replaced by TDB which first turns and then moves the crawler.

The FRB is executed until the data crawler has moved a predetermined number of times, i.e. $9 * w$ times, where w is the width of a vehicle. This width is automatically scaled by aircraft altitude so that w at 600ft is half w at 300ft (Roberts and Howard, 1999). FRB is iteratively evaluated until the crawler executes the predetermined number of moves. In future research the number of moves will not be predetermined but will also be allowed to evolve.

The number of image clues that could be flagged was limited to 50. When this limit was exceeded, the FRB aborted and returned the clues found so far.

3.2 Turn Decision Branch (TDB)

TDB first decides on a direction of travel and then moves the crawler in this direction. Terminals μ_{11}^C, σ_{11}^C are averages and standard deviations obtained from the pixel values in an eleven pixel square window centred on the crawler. Labels C, N, E, S, W stand for centre, north, east, south and west locations as shown in Figure 2. The window size is automatically scaled according to altitude.

TDB is always evaluated four times, once for each permutation of the order of the terminals at directions N, E, S, and W, and this produces four outputs. In the current implementation, the chosen data crawler direction is given by the permutation returning the largest output and follows certain rules.

The MOVE memory is updated after every data crawler move, i.e. when the FRB invokes M or TDB. This memory records the local turns of the crawler: forward, right, back, left (F, R, B, L) rather than the absolute directions N, E, S, and W. Pixel statistics are also recorded and so the MOVE memory consists of:

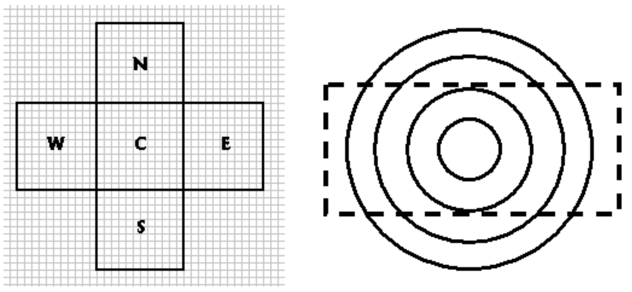

Figure 2: Left: TDB terminals: mean and standard deviation computed over square areas above, below, to the right, to the left, and centred on the pixel. These are the 'sensors' of the data crawler. Right: Four concentric pixel rings centred on a vehicle with diameters: 0.5, 1.0, 1.5 and 2.0 vehicle widths.

1. last 10 directions, e.g. F,R,B,B,F,F,B,L,R,L is stored as [1,2,3,3,1,1,3,4,2,4];
2. last 10 μ_{11}^C;
3. last 10 σ_{11}^C;

At the start of travel, the direction memory locations are initialized to 0 and all ten values in each statistic memory location are set to the initial μ_{11}^C and σ_{11}^C.

3.3 Mark Decision Branch (MDB)

MDB determines whether a flag should be left at the current pixel position in the trail to indicate the presence of an 'image clue'. If the MDB returns a positive number it deposits a flag. This decision is based on pixel data and trail data. MDB acts as an 'IF' statement in the FRB. Placing a flag executes a THEN subtree, otherwise an ELSE subtree is executed. The executed portion is returned to FRB.

Terminals μ_{disk} and σ_{disk} in Table 2 are averages and standard deviations that are calculated over a small disk centred on the current pixel. The disk diameter is half the width of a car and corresponds to the smallest disk on the right of Figure 2. This disk is scaled by aircraft altitude, and it is distorted into an ellipse when the image is at perspective due to aircraft roll.

The FLAG memory is updated following the call to MDB. This memory consists of:

1. last 10 MDB results, e.g. [-1,-1,1,1,1,1,-1,1,-1,-1];

2. last 10 μ_{disk};
3. last 10 σ_{disk};

Here -1 stands for 'no flag', and 1 stands for 'flag set'. Initially, all values in the vector of results are set to -1 and in the other two memory vectors are set to the initial μ_{disk} and σ_{disk} respectively.

WRITEMEM and READMEM write to and read from the indexed WORKING memory, which consists of three locations or slots. All three locations are initialized to 0.0 before the crawl. Note that successive calls to MDB from FRB will not move the data crawler. The result of the repeated calls, however, can differ from one another because the memory states can change between calls. Note also that the data crawler may revisit a pixel and change its decision about flag placement.

3.4 Second Result Branch (SRB)

SRB returns a real numbered value that can be either positive (target is present) or negative (no target is present). The SRB is only executed after the FRB has completed.

The FRB uses a 2D square map to store the deposited flags. The map is initialized to 0 values and flag locations are indicated by 1 values. The suspicious pixel which is the starting point of the crawl, is at the centre of this map. If the map stores fewer than a threshold number of flags, $T_F = 4$, then the suspicious pixel is labelled 'negative' (no target present), and the SRB is not invoked. If the map stores at least T_F flags, then the SRB processes statistical measures based on the distribution of the flags.

The ten flag based terminals in Table 2 require computation of the centre of mass of the flags. They are arbitrarily based on statistical measures over the vector of distances between the centre of mass and each flag. All ten statistics are positive in value:

1. the number of flags;
2. the distance from the centre of mass to the furthest flag;
3. the longest distance between any two flags;
4. the average distance between any two flags;
5. the standard deviation in distance between any two flags;
6. the average distance between the centre of mass and each flag;
7. the standard deviation in distance between the centre of mass and each flag;
8. the degree of asymmetry of the distance distribution (skewness);
9. the relative shape of the distribution compared with a normal distribution (kurtosis);
10. the co-variance after the vector is sorted into ascending order and halved to form two sub-distributions.

Textural statistics over a wide area are input to SRB so that the branch can form its own image segmentation. In this way, the SRB can give different detections for the same flag distribution appearing in different textural contexts. For example, a horse in a rural area may receive the same flag distribution as a vehicle in an urban area, but the SRB could discriminate the horse as a false alarm, whilst detecting the vehicle, due to the differences in rural and urban textures (Roberts and Howard, 1999). These textural statistics are based on a co-occurrence matrix and are taken over an area of 5 car-lengths square.

Table 3: GP parameters.

Parameter	**Setting**
kill tournament	size 2 for steady-state GP
breed tournament	size 4 for steady-state GP
regeneration	90% x-over, 0% clone, 10% truncation mutation
population	500
max generations	50
max branch size	1000 nodes

4 GP implementation

The GP run parameters are given in Table 3. Crossover was branch typed, meaning that it could only exchange genetic material between like branches, e.g. an FRB only with another FRB. Each branch was assigned a probability to participate in crossover. The FRB had a probability of 40% and the other three branches had a probability of 20%. Truncation mutation selects any node (and the associated subtree) and replaces it with a terminal from the relevant terminal set.

The suspicious points detected by the (Roberts and Howard, 1999) scheme constituted the fitness cases. These points were known to be 'true positives' (TP) or 'false positives' (FP), i.e. false alarms. The fitness

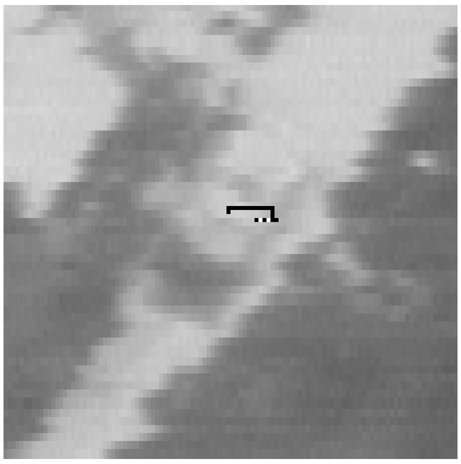
Figure 3: Trail terminating at the right of a vehicle.

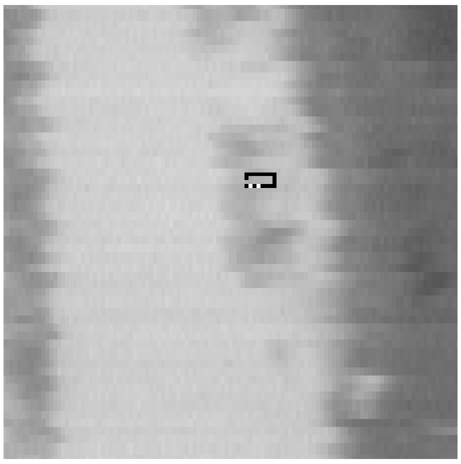
Figure 4: Trail terminating at the left of a vehicle.

measure was computed once all fitness cases had been processed:

$$\text{fitness} = \frac{\alpha TP}{\beta FP + \text{nVehicles}}$$

The parameters α and β balance the importance of TP relative to FP. These parameters were optimized to minimize FP whilst retaining near-maximal TP.

5 Experimental Results

The scheme was implemented to process previously detected vehicles and false alarms in airborne reconnaissance IRLS imagery (Roberts and Howard, 1999). The vehicles appear in many sizes and orientations, in many perspectives and thermal states, next to various objects (such as buildings) which cast thermal shadows onto the vehicles, and in many environments and weather conditions. Hence, the vehicle detection task in these operational images is extremely challenging.

This section presents examples of the trails produced by a data crawler evolved with $\alpha = 2.6$ and $\beta = 1.0$. The trails are drawn on sub-images cropped from IRLS imagery with approximate dimensions of $3000 \times 10,000$ pixels. The end of each trail is indicated with two dots, except when the trail is iterative.

Figures 3 and 4 show that the crawler can make a similar trail on different vehicles. Each trail starts on the vehicle's roof because this was the starting point as detected by the (Roberts and Howard, 1999) scheme. Interestingly however, each trail terminates at the vehicle's side, even though the vehicles are oriented differently.

Generally, when the crawler is applied to the *same* vehicle, but with different starting points, it can take different routes but still detect the vehicle. Figure 5 displays a case where the crawler homed in on a vehicle's wind-screen.

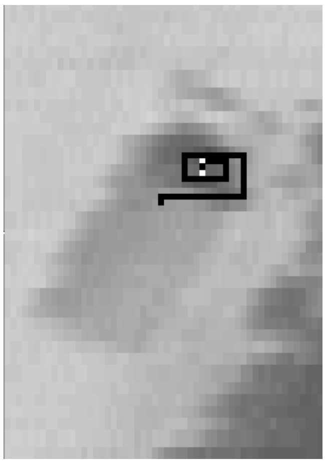
Figure 5: Spiral trail on a vehicle.

The power of the data crawler is illustrated in Figures 6 and 7, where false alarms are rejected due to contextual information. In other words, the 'false positives' as detected by the (Roberts and Howard, 1999) scheme are correctly converted to 'true negatives'. In Figure 6, the starting point could resemble some feature of a vehicle, but crawling onto an open roof dismissed this hypothesis. Similarly, a more complicated trail in Figure 7 rejects a grass verge by crawling onto an open carriageway. The grass verge was probably initially detected as a false alarm because it has the width of a car. This sub-image clearly shows the jitter in the IRLS imagery which hinders the detection task. Experiments reduced the false alarm rate typi-

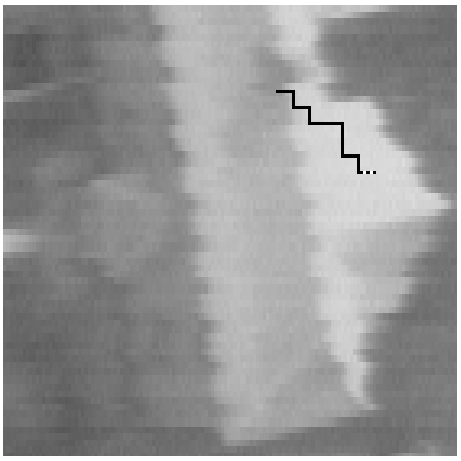

Figure 6: A potential vehicle feature on a roof is correctly rejected.

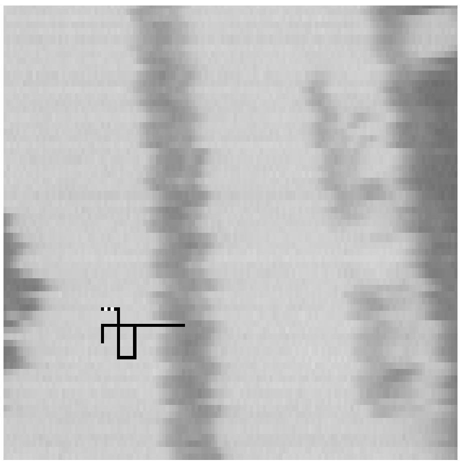

Figure 7: Trail commencing on a grass verge between carriageways. Note the two cars on the right.

cally by 30%.

Other false alarms are rejected not by wandering trails, but instead by tight trails which stay close to the starting point, as shown in Figures 8 to 10. In these cases, the starting point is within the confines of an object which is too small to be a target, and thus there is no need to explore the object's surroundings. Note that the ends of these trails are not shown because the trails are iterative, with the crawler retracing its steps or performing a cyclic motion.

Systematic flag formations to characterize object detections are not visibly evident in the crawler's trails. For example, the false alarms are not characterized by a typical flag distribution pattern. This is probably because the flag placement, and indeed the trail itself,

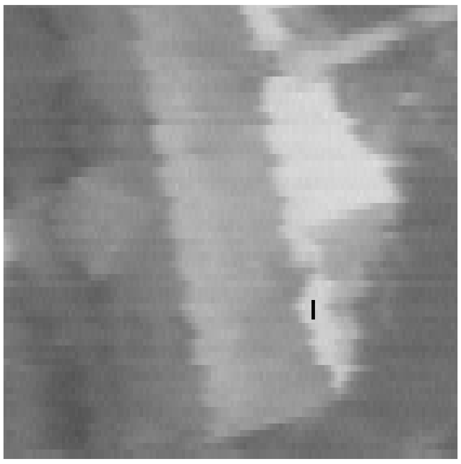

Figure 8: Iterative trail on an obstructed section of roof.

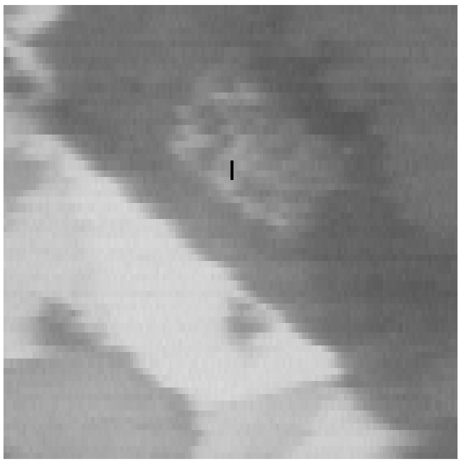

Figure 9: Iterative trail on a garden plot.

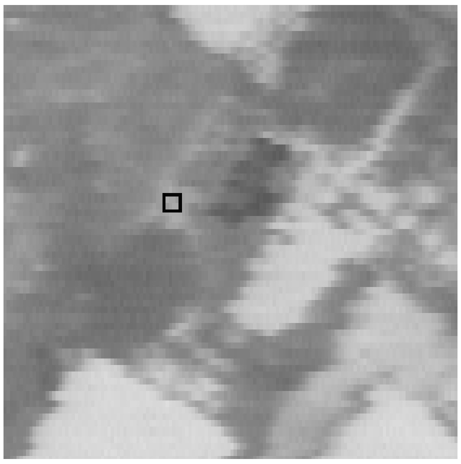

Figure 10: Cyclic trail in a rural driveway near a tree and out-buildings.

depends on the particular 'image clues' in an object's vicinity.

However, some typical flag distribution patterns are evident. There is a tendency for the crawler to place flags with a near-equal distribution along the trail, and to prefer placements on turning points. More strikingly, the last few steps in each trail are often flagged. The crawler governs the length of its trail by iteratively retracing its final steps to effectively time-out the FRB. Hence, these final steps are often *repeatedly* flagged.

Recall that at least $T_F = 4$ flags had to be set for the SRB to be executed, and that fewer flags gave a default negative detection (Section 3.4). Experiments found that the number of individuals that place at least T_F flags increased during each evolution run. This suggests that individuals could not use the number of flags alone to detect the vehicles, by simply invoking SRB to output a positive value. This also suggests that SRB was employed to manipulate the flag distribution statistics in a more sophisticated way.

For example, each step in the iterative trails in Figures 8 and 9 are flagged, thus yielding short distances for the flag distribution statistics. The SRB could then interpret these statistics as an indicator to reject these false alarms.

6 Conclusions

This paper describes a data crawler to improve target detection by exploring the context in which candidate targets appear. The crawler initially processes *close-up* views to "sense" an object's surroundings and to flag where it becomes aroused. The distribution of these flags then represents a *distant* view of the object, where contextual information has necessarily been greatly reduced. The crawler then indicates whether the object is a required target or a false alarm. All aspects of the crawler's design are evolved using GP with fixed architecture ADFs.

The data crawler improved the performance of a preliminary target detection scheme, by typically reducing the false alarm rate by 30% whilst retaining most of the actual targets.

The present scheme is computationally demanding, due to the calculation of pixel statistics every time the crawler moves. Computation speed may be improved by allowing the crawler to jump, and indeed this may be more in-keeping with the way that a scene is scanned in animal vision. Other advancements could explore iterations between the SRB and FRB to reinforce detection decisions, and "colonies" of crawlers could participate via pheromone trails.

References

[Andre, 1994] David Andre (1994). Automatically Defined Features: The Simultaneous Evolution of 2-Dimensional Feature Detectors and an Algorithm for Using Them. In Kenneth E. Kinnear, Jr. (ed), *Advances in Genetic Programming, 477–494*, MIT Press.

[Benson, 2000] Karl A Benson (2000). Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection within SAR Imagery. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00, 1543–1549*, IEEE Press.

[Daida et al., 1996] Jason M. Daida, R. G. Onstott, T. F. Bersano-Begey, S. J. Ross and J. F. Vesecky (1996). Ice Roughness Classification and ERS SAR Imagery of Arctic Sea Ice: Evaluation of Feature-Extraction Algorithms by Genetic Programming. In *Proceedings of the 1996 International Geoscience and Remote Sensing Symposium, 1520–1522*, IEEE Press, Washington.

[Harris and Buxton, 1996] Christopher Harris and Bernard Buxton (1996). Evolving Edge Detectors with Genetic Programming. In Koza, Goldberg, Fogel and Riolo, *Genetic Programming 1996: Proceedings of the First Annual Conference, 309–315*, MIT Press.

[Howard et al., 1999] Daniel Howard, Simon C. Roberts and Richard Brankin (1999). Evolution of Ship Detectors for Satellite SAR Imagery. In Poli, Nordin, Langdon and Fogarty (eds), *Genetic Programming, Proceedings of EuroGP 1999, 135–148*, Lecture Notes in Computer Science 1598, Springer-Verlag.

[Howard and Roberts, 1999] Daniel Howard and Simon C. Roberts (1999). A Staged Genetic Programming Strategy for Image Analysis. In Banzhaf, Daida, Eiben, Garzon, Honavar, Jakiela and Smith (eds), *Proceedings of the Genetic and Evolutionary Computation Conference, 1047–1052*, Morgan Kaufmann.

[Howard and Roberts, 1999b] Daniel Howard and Simon C. Roberts (1999). Evolving Object Detectors for Infrared Imagery: a Comparison of Textural Analysis Against Simple Statistics. In Miettinen, Mäkelä and Toivanen (eds), *Proceedings of EUROGEN99 - Short Course on Evolutionary Algorithms in Engineering and Computer Science, 79–86*, Dept. of Mathematical I. T. Collections No. A 2/1999, University of Jyväskylä, Finland, ISBN 951-39-0473-3.

[Howard et al., 2002] Daniel Howard, Simon C. Roberts and Conor Ryan (2002). The BORU Data Crawler for Object Detection Tasks in Machine Vision. *Applications of Evolutionary Computing: European EC Workshop 2002, Kinsale, Ireland, 222–232*, Lecture Notes in Computer Science 2279, Springer-Verlag.

[Johnson et al., 1994] Michael Patrick Johnson, Pattie Maes and Trevor Darrell (1994). Evolving Visual Routines. In Rodney A. Brooks and Pattie Maes (eds), *ARTIFICIAL LIFE IV, Proceedings of the fourth International Workshop on the Synthesis and Simulation of Living Systems, 198–209*, MIT Press.

[Koza, 1992] John R. Koza (1992) Genetic Programming, MIT Press.

[Koza, 1994] John R. Koza (1994) Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press.

[Poli, 1996] Riccardo Poli (1996). Genetic Programming for Image Analysis. In Koza, Goldberg, Fogel and Riolo (eds), *Genetic Programming 1996: Proceedings of the First Annual Conference, 363–368*, MIT Press.

[Roberts and Howard, 1999] Simon C. Roberts and Daniel Howard (1999). Evolution of Vehicle Detectors for Infrared Linescan Imagery. In Poli, Voigt, Cagnoni, Corne, Smith and Fogarty, *Evolutionary Image Analysis, Signal Processing and Telecommunications: First European Workshop, EvoIASP 1999 and EuroEcTel 1999, 110–125*, Lecture Notes in Computer Science 1596, Springer-Verlag.

[Roberts and Howard, 2000] Simon C. Roberts and Daniel Howard (2000). Genetic Programming for Image Analysis: Orientation Detection. In Whitley, Goldberg, Cantu-Paz, Spector, Parmee and Beyer, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), 651–657*, Morgan Kaufmann.

[Roberts et al., 2001] Simon C. Roberts, Daniel Howard and John R. Koza (2001). Evolving Modules in Genetic Programming by Subtree Encapsulation. In Miller, Tomassini, Lanzi, Ryan, Tettamanzi and Langdon (eds), *Genetic Programming, Proceedings of EuroGP 2001, 160–175*, Lecture Notes in Computer Science 2038, Springer-Verlag.

[Tackett, 1993] Walter Alden Tackett (1993). Genetic Programming for Feature Discovery and Image Discrimination. In Stephanie Forrest (ed), *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, 303–309*, Morgan Kaufmann.

[Teller and Veloso, 1996] Astro Teller and Manuela Veloso (1996). PADO: A New Learning Architecture for Object Recognition. In Katsushi Ikeuchi and Manuela Veloso, *Symbolic Visual Learning*, Oxford University Press.