

# Learning Visual Feature Detectors for Obstacle Avoidance using Genetic Programming

Andrew Marek      William D. Smart  
Department of Computer Science and Engineering  
Washington University  
St. Louis, MO 63130  
United States  
{ajm2,wds}@cse.wustl.edu

Martin C. Martin  
MIT Artificial Intelligence Laboratory  
200 Technology Square, NE43-933  
Cambridge, MA 02139  
United States  
martin@metahuman.org

## Abstract

*In this paper, we describe the use of Genetic Programming (GP) techniques to learn a visual feature detection for a mobile robot navigation task. We provide experimental results across a number of different environments, each with different characteristics, and draw conclusions about the performance of the learned feature detector. We also explore the utility of seeding the initial population with a previously evolved individual, and discuss the performance of the resulting individuals.*

## 1. Introduction

Feature extraction from images is a difficult task. Assumptions about the nature of the environments, and how they appear in images often prove to be incorrect, causing the performance of feature extractors to be lowered. In this paper, we discuss our experiences using Genetic Programming (GP) techniques [7] to evolve a visual feature detection algorithm to be used for obstacle avoidance on a mobile robot.

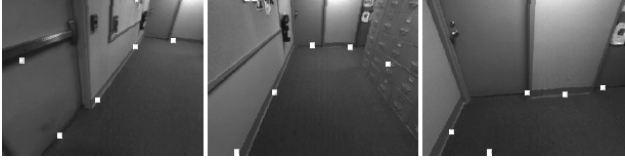
Since such algorithms are often fine-tuned to work on their particular training set, we have performed a series of experiments across different environments, each with its own distinctive characteristics and challenges. By taking individuals evolved to deal with one environment and running them in each of the other environments, we hope to gain a better understanding of what is needed in a training set for optimal performance in this particular task. We also explore the use of a previously-evolved seed individual in the initial population in order to ensure evolutionary success and high fitness levels.

The work reported here is a direct extension of Martin's [9], and attempts to test how well the results previously re-

ported generalize across several environments. We begin by discussing the particular domain that we are interested in developing feature detectors for, robot obstacle avoidance. We then briefly summarize some related work in the area of feature detection. Next, we describe our experimental setup, and then give the results of our experiments. Finally, we offer some conclusions from this work and discuss the directions in which we would like to take it.

## 2. Genetic Programming

Genetic Programming (GP) is a Genetic Algorithm [2] where the individuals are programs, represented as parse trees. The set of nodes used for a given problem is chosen by the programmer for the problem domain. In GP parlance, nodes with no children (leaf nodes) are dubbed terminals, those with children (internal nodes) are called functions. Trees are created by randomly choosing a node for the root, then choosing random nodes for each of its arguments (children), and similarly for their arguments, etc. A fixed number of trees are randomly created this way, and each one evaluated, and assigned a real number that measures how well it solves a given problem. The assigned number is called the fitness. Next, individuals are randomly selected, with fitter individuals more likely to be chosen. The selected individuals are either copied verbatim into the new population (replication), or undergo crossover. Crossover works by selecting a random node in each of two parents and swapping the subtrees rooted at those nodes, then inserting both of these new individuals into the population. Once the required number of individuals are created in this way, their fitness is measured and the process repeats for a fixed number of generations. Typically, the individual with the best fitness is designated as the result of the GP run. Creation and crossover are constrained so that resulting trees are viable programs.



**Figure 1. Environment and detected features from Martin [9].**

GP essentially performs a search through the space of possible programs. It is more computationally efficient than an exhaustive search, but at the expense of the quality of the final solution. GP is not guaranteed to find the optimal solution, but often succeeds in finding a good one. GP is also able to overcome some of the local optima problems associated with local search techniques.

### 3. Robot Obstacle Avoidance

Martin [9] used GP methods to learn algorithms that extracted the height of the the lowest non-floor pixel in a given column of an image. In the office environments in which the work was done, this corresponded to the floor/wall interface, and served as a relative distance measure to the wall. The positions of these lowest non-floor pixels were then used as input to a simple mobile robot navigation algorithm, inspired by the work of Horswill [4]. The basic idea of this algorithm is to steer the robot towards the area that is most open. Since the height of the lowest non-floor pixel corresponds roughly to distance, this means steering the robot towards the highest of the features identified. This system resulted in a surprisingly robust navigation behavior. Figure 1 shows examples of the environment and the identified features. In all of these experiments, fitness was measured by the distance between the actual lowest non-floor pixel and the position in which the program decided it should be.

Martin was able to successfully produce individuals which were able to identify the floor/wall interface to within 10 pixels approximately 85% of the time. This performance level was achieved despite burned out lights, shadows from the robot, imaging artifacts, and moiré patterns in the image. The results from these experiments were more than good enough for the navigation algorithms to use successfully.

Martin also experimented with seeding the initial population by inserting a poorly performing hand-written program into the initial population. These experiments are described in detail by Martin [9], and form the basis of the work reported here. Based on Martin’s observations, we are interested in looking in more detail at the effects of an

evolved (rather than hand-written) seed individual, at the effects of training and testing in different environments and, in the long run, at using GP for learning programs for more general visual detection problems.

## 4. Related Work

Several researchers have applied GP to finding things in images. The most directly related to the work proposed here is by Martin, and is described above.

Harvey, Husbands and Cliff [3] use GP to evolve the control and morphology of an extremely simple vision system for a gantry-based mobile robot. The system consists of three receptive fields, and their position is learned by GP so that they can detect a small number of simple targets. Lut-ton and Martinez [8] investigate the use of GP to learn programs that extract geometric primitives from images. Interestingly, they found that the evolved programs were complimentary to the widely-used Hough Transform [5]. For simple primitives, up to three parameters, the evolved programs perform poorly, and the Hough Transform is the method of choice. For more complex primitives, the memory and data requirements of the Hough Transform become unreasonable, but the evolved programs produce reasonable results.

Not using genetic programming, but similar in spirit to the work proposed here, Draper, Bins and Baek [1] use reinforcement learning techniques to learn good sequences of image operators for detecting houses in aerial photographs. Other AI researchers have also applied planning techniques to induce good sequences of operators for image processing tasks [6].

## 5. Experiments

In this section, we describe the environments in which we collected training and testing data from and outline our experimental setup.

### 5.1. Environments

We gathered data from four different indoor environments, examples of which are shown in figure 2. The details of these environments are as follows:

**A1** Office/corridor environment with sharp shadows from strong sunlight coming through open doors and windows. White walls with dark molding at the bottom. Carpet is lighter than the molding, but darker than the walls.

**A2** Same environment as A1, but without shadows.

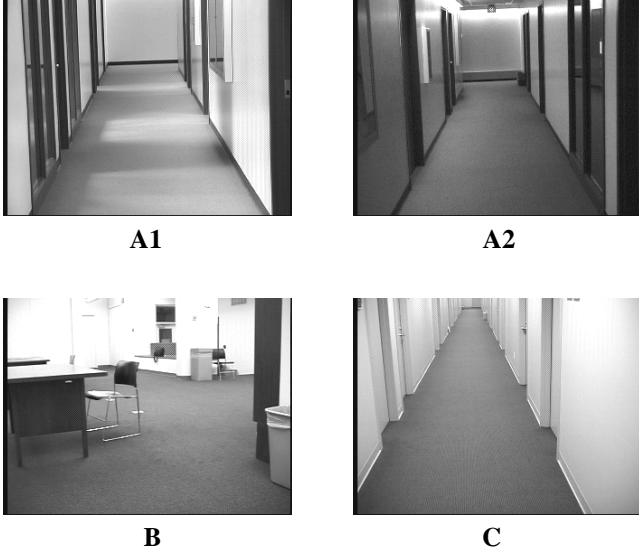


Figure 2. The four test environments.

- B** Office conference room environment, with sharp shadows created by overhead lights and furniture. Light-colored walls, with dark carpet that is the same color as the molding.
- C** Library environment, with no shadows. White walls, with white molding and darker carpet.
- all** Frames from all four data sets combined. This data set consists of 70 frames, randomly drawn from the other four data sets.

We took 350 frames of video while moving through each of these environments, and used every 5th frame to construct the training sets. Each of these frames has ground truth information, corresponding to the floor/wall interface in 6 pre-specified columns, added to it by a human. To evaluate fitness, we compare this ground truth information for all of the images in a data set with the predicted positions, for a total of 420 fitness cases. If the predicted positions are within  $\pm 10$  pixels of the ground truth, the prediction is classified as correct, with total fitness being the number of correct predictions over the total number of fitness cases.

The combined data set (**all**) uses a total of 70 frames, drawn uniformly from all four training data sets, and evaluates fitness in the same way as the four individual data sets.

## 5.2. Experimental Setup

We want to be able to identify the lowest non-floor pixel in a given column of an image. For our experiments, we selected six pre-specified, evenly-spaced columns in each image. In Martin’s work these six columns corresponded

Node List	
root	iterate-up,iterate-down
window sizes	r22, r23, r32, r33, r24, r42, r44, r55, r26, r62, r66, r77, r28, r82, r38, r83, r88
arithmetic	*, +, /, -, square, 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); firstrect (1 if this is the first rectangle of the iteration, 0 otherwise) x and y (the center of the rectangle in pixels)
flow control	prog2; prog3; break (halts the execution of the branch, returning immediately without any more iterations), if-le ( $\leq$ operator)
registers	set-a ... set-e, read-a ... read-e
image statistics	average & average-of-squared over the window, raw, truncated median, median corner, Sobel magnitude, and four directional Moravec interest operators

Table 1. List of available functions and terminals.

directly to the number of sonar sensors within the camera’s field of view. In our work, however, the six columns are specified by hand and vary from image to image while maintaining as close to an even distribution as possible. In this setup, only vertical iteration was used with the iterate node taking three (learned) arguments: the window size (window width = column width), horizontal location, and the sub-program to execute at every step as the window moves vertically along the column. This setup produced individuals which achieved fitness levels exceeding 85% on Martin’s original training and test data, and similar levels on our own experiments. Fitness was evaluated as described above in section 5.1

Our experiments used a population size of 4,000 individuals. These individuals consisted of a set of nodes, or functions and terminals. The full list of node types is summarized in table 1. In addition to the iterate and rectangle sizes (r22 = 2x2 window), the list includes standard arithmetic operators, various parameters of the image, flow control, nodes to set and read five floating point registers, and the average and average of squared value of common image operators. The registers were inspired by Teller et al. [10].

Train	Test Data Set				
	A1	A2	B	C	all
A1	<u>81%</u>	62%	36%	39%	56%
A2	63%	<u>80%</u>	21%	53%	57%
B	74%	74%	<u>83%</u>	74%	77%
C	12%	35%	39%	<u>86%</u>	40%
all	78%	78%	77%	<u>81%</u>	79%

**Table 2. Fitness of best individuals after 50 generations of evolution. Best fitness for each test set is underlined.**

Martin experienced little improvement after the 50th generation [9]. For this reason, we limit ourselves to 50 generations of evolution. Running for longer, with more generations, will almost certainly improve performance, but the performance after 50 generations seems to be a reliable indicator of final expected performance. Individuals are chosen using tournament selection with a size of 7. Genetic operators include crossover (90% rate), with a 90% probability of selecting a function and a 10% probability of a terminal. In order to prevent code bloat, individuals created through crossover are limited in size to 1000 nodes. Reproduction, or replication, also is used with entire, unchanged, individuals being moved to the next generation at a rate of 10%. Mutation is not used, since, empirically, crossover seems to introduce enough disruption onto the population.

## 6. Experimental Results

We begin this section by discussing the performance of the GP system across the environments described in the previous section. During the course of running these experiments, we noticed an interesting bi-modal distribution in the final fitness of the best individuals. Either the best individual was very good or very bad. There were no “in-between” cases where the best individual had a middling fitness. We discuss this further below.

### 6.1. Performance Across Environments

For each of the five training data sets, we performed ten runs of GP, evaluating each run on its own data set. We then selected the best individual from all of these runs, and evaluated it on the other data sets. The results for the cross-environment experiments are shown in table 2. Individuals evolved for a particular environment do better there than in any of the others. Additionally, higher fitness was observed across environments that were similar (for example **A1** and **A2**) than across those that were significantly different (such as **A1** and **B**). This behavior is not surprising, and suggests

that the best individuals are using features that are unique to their training environment. As expected, the results on the combined data sets are approximately the average of those on the individual ones.

One interesting feature of these results is that individuals tested on data set **B** uniformly perform poorly (with the exception of those also trained on **B**). However, individuals trained on **B**, uniformly perform well on other test sets. Data set **B** has less rigid structure and is more varied than the other environments (see figure 2). These characteristics suggest that less structured environments should be preferred for training, since they will tend to produce more general-purpose individuals. How to determine which environments meet this criterion, however, is not an easy problem.

Finally, individuals trained on a combination of all of the data performed well on all of the data sets. It is interesting to note that the fitness of these individuals is not much better (in most cases) than the fitness of the individuals trained on data set **B**. It seems that the difficulty and diversity of this training set is almost enough on its own to produce a robust algorithm that works across environments. Examples of the features detected by the best individuals are shown in figure 3.

Indeed, the fitness on data set **B** and on the combined data set demonstrate that individuals can generalize well by making sure the training data has two important characteristics: diversity and difficulty.

### 6.2. Bimodal Performance Distribution

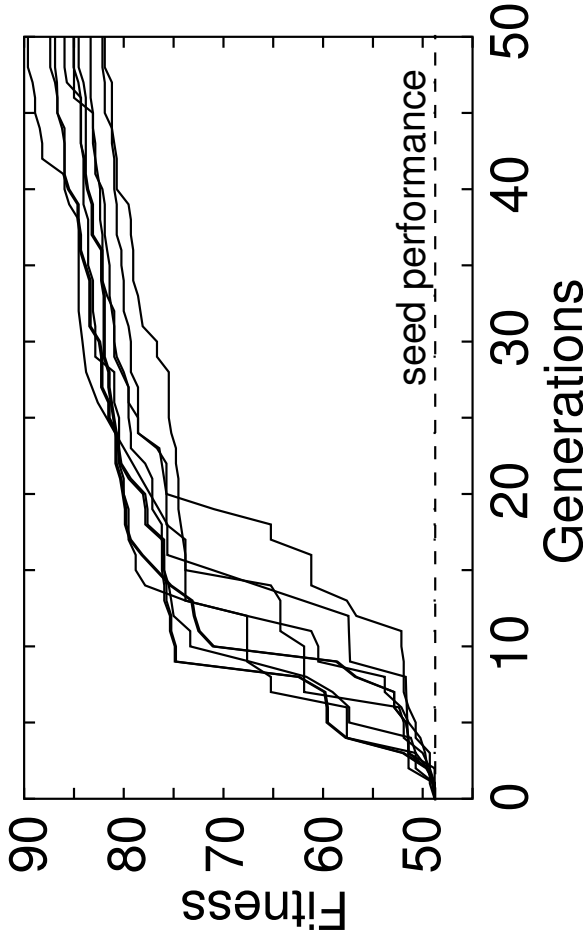
The best individual from each experiment fell into one of two distinct groups, one with high fitness (70–90%), and one that failed to get very far past a fitness of 50%. Moreover, in all experiments there is a very definite point, over the space of a few generations, where the fitness moves from the lower mode to the upper one. In all five of the data sets, if the fitness of the best individual did not reach 50% by generation 24 (in all experiments, except one that transitioned after generation 42), the run was doomed to remain in the lower mode. This grouping is very apparent in figure 4, which shows the fitness of all individuals from one experimental run, for each of the data sets.

This behavior, where the fate of an experimental run is sealed by generation 24, suggests that an individual must learn some key operator, function, procedure, or combination that is instrumental to its future success within those first 24 generations. Given the way that GP works, this is an understandable behavior. If we can find out what this vital piece of the algorithm is, we could insert it into the population from the beginning. This algorithm piece should greatly increase the chances of a successful experiment, possibly even guaranteeing them.



	Subtree 1		Subtree 2	
	With	Without	With	Without
Mean	75%	47%	52%	42%
Variance	2.34%	4.54%	6.00%	5.28%
t-statistic	3.35		0.88	
confidence	99.5%		80%	

**Table 3. Fitness with and without the seed subtree for two subtrees. The t-statistic is for the t-test of the hypothesis that the fitness with the subtree is greater than the fitness without. Confidence is the corresponding confidence level at which this hypothesis holds.**



**Figure 5. Ten best individuals from a seeded run in A1.**

set **A1** and found that in generations after large fitness increases, certain function clusters, or subtrees, would be present and would persist throughout the evolutionary run. This persistence seemed to indicate that these subtrees are important to feature detection. Therefore, we decided to take these subtrees and include them in the original function set, so that they would be available to an evolutionary run from the start. We identified two key subtrees in this manner and executed ten runs with the first subtree and ten runs without. The results of these experiments are shown in table 3.

Runs with subtree 1 performed significantly better than those without it. With 99.5% confidence, we can say that the presence of of subtree 1 helps increase fitness, at least in our domain.

The results from the experiments with the second subtree are also shown. Runs with the subtree averaged a fitness of 52%, compared to 42% without it. While we can say that the presence of this function cluster helped increase fitness significantly, we cannot say it with as much force, because statistically we can only be 80% confident.

## 7. Conclusions and Future Work

In this paper, we have presented some preliminary results of using GP to learn a visual feature detector for mobile robot navigation domains. We discussed the results of experiments across several environments, and analyzed a bimodal learning behavior, which resulted in the final performance either being very good or very bad. The resulting individuals have been used to supply input for a mobile robot performing a navigation task, and we are currently looking at how the quality of the individual affects the quality of the navigation.

Although this paper deals with a particular set of experiments in a particular domain, we believe that it sheds some

light on GP in general. We found that the use of an evolved seed can have a great effect on the final performance level of the system. We believe that evolved seeds have the benefit of being easily manipulated by the GP system, as well as requiring no knowledge of how to code up an individual from scratch. One of the things that we plan to investigate in future work is the question of how the quality (in terms of size, efficiency, *etc.*) of the initial seeds affects the final performance of the learned individuals.

In addition, we have demonstrated that evolved subtrees, or terminal and function clusters, can increase GP performance significantly. An interesting offshoot of this work will be to see if GP finds the same sort of image operations, or combinations of operations that a human expert would use, or if it comes up with radically different solutions.

We are currently looking at extending this work to learn programs to identify more general features in images. We are interested in investigating the types of features that can be detected with this approach, the performance that we can achieve (can we do better than a human expert?), and if we can use this approach to learn programs that will fuse data from more than one sensor.

## References

- [1] B. A. Draper, J. Bins, and K. Baek. ADORE: Adaptive object recognition. *Videre: Journal of Computer Vision Research*, 1(4), 2000.
- [2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [3] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, pages 392–401, Cambridge, MA, 1994. MIT Press.
- [4] I. Horswill. Polly: A vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 1993.
- [5] P. V. H. Hough. A new method and means for recognizing complex pattern, 1962. U.S. Patent 30690653.
- [6] X. Jiang and H. Binke. Vision planner for an intelligent multisensory vision system. In *Automatic Object Recognition IV*, pages 226–237. 1994.
- [7] J. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.
- [8] E. Lutton and P. Marintez. A genetic algorithm with sharing for the detection of 2d geometric primitives in images. In J.-M. Alliot, E. Lutton, E. Ronald, and D. Snyers, editors, *Artificial Evolution*, volume 1063 of *Lecture Notes in Computer Science*, pages 287–303. Springer, Berlin, Germany, 1995.
- [9] M. C. Martin. *The Simulated Evolution of Robot Perception*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2001.
- [10] A. Teller and M. Veloso. PADO: Learning tree-structured algorithms for orchestration into an object recognition system. Technical Report CMU-CS-95-101, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1995.