Report Number: WUCSE-2004-22

2004-04-19

Learning Feature Detectors Using Genetic Programming With Multiple Sensors

Andrew Marek

In this thesis, we describe the use of Genetic Programming (GP) to learn obstacle detectors to be used for obstacle avoidance on a mobile robot. The first group of experiments focus on learning visual feature detectors for this 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 and the training data used to learn such detectors. We also explore the utility of seeding the initial population with previously evolved individuals and subtrees, and discuss the performance of the resulting individuals. We then... **Read complete abstract on page 2.**

Learning Feature Detectors Using Genetic Programming With Multiple Sensors

Andrew Marek

Complete Abstract:

In this thesis, we describe the use of Genetic Programming (GP) to learn obstacle detectors to be used for obstacle avoidance on a mobile robot. The first group of experiments focus on learning visual feature detectors for this 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 and the training data used to learn such detectors. We also explore the utility of seeding the initial population with previously evolved individuals and subtrees, and discuss the performance of the resulting individuals. We then include sensory data from a laser range-finder and a camera and discuss the performance of resulting individuals as we use just laser data, just image data, and both in combination.

2004-22

Learning Feature Detectors Using Genetic Programming With Multiple Sensors, Master's Thesis, May 2004

Authors: Marek, Andrew

April 19, 2004

Abstract: In this thesis, we describe the use of Genetic Programming (GP) to learn obstacle detectors to be used for obstacle avoidance on a mobile robot. The first group of experiments focus on learning visual feature detectors for this 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 and the training data used to learn such detectors. We also explore the utility of seeding the initial population with previously evolved individuals and subtrees, and discuss the performance of the resulting individuals. We then include sensory data from a laser range-finder and a camera and discuss the performance of resulting individuals as we use just laser data, just image data, and both in combination.

SEVER INSTITUTE OF TECHNOLOGY

MASTER OF SCIENCE DEGREE

THESIS ACCEPTANCE

(To be the first page of each copy of the thesis)

DATE: April 19, 2004

STUDENT'S NAME: Andrew J. Marek

This student's thesis, entitled Learning Feature Detectors Using Genetic Programming With Multiple Sensors has been examined by the undersigned committee of five faculty members and has received full approval for acceptance in partial fulfillment of the requirements for the degree Master of Science.

APPROVAL: _____ Chairman

_____

_____

_____

_____

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LEARNING FEATURE DETECTORS USING GENETIC PROGRAMMING WITH MULTIPLE SENSORS

by

Andrew J. Marek, B.S.

Prepared under the direction of Dr. William D. Smart

A thesis presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Master of Science

May, 2004

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

LEARNING FEATURE DETECTORS USING GENETIC PROGRAMMING WITH MULTIPLE SENSORS

by Andrew J. Marek

ADVISOR: Dr. William D. Smart

May, 2004
Saint Louis, Missouri

In this thesis, we describe the use of Genetic Programming (GP) to learn obstacle detectors to be used for obstacle avoidance on a mobile robot. The first group of experiments focus on learning visual feature detectors for this 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 and the training data used to learn such detectors. We also explore the utility of seeding the initial population with previously evolved individuals and subtrees, and discuss the performance of the resulting individuals. We then include sensory data from a laser range-finder and a camera and discuss the performance of resulting individuals as we use just laser data, just image data, and both in combination.

to my parents

Contents

List of Tables

List of Figures

Acknowledgments

Thanks to the my parents for their unwavering support and love. I hope to someday thank you enough for all that you have given me. Thanks to Marisa for putting up with me through it all. You are the source of so much happiness and laughter in my life. Many thanks to Bill Smart, my advisor. You were always there to give me a kick in the pants when I needed it, and your ideas and advice were invaluable. Thanks to Martin Martin, our indispensible launch point and GP guru for all of this. Thanks to Christian Gagne, the Open BEAGLE creator, whose framework and collaboration was key to the endeavors pursued here.

Andrew J. Marek

Washington University in Saint Louis
May 2004

Chapter 1

Overview

1.1 Introduction

Feature extraction from images and laser data is a difficult task. Assumptions about the nature of the environments, and how they appear in images and laser scans often prove to be incorrect, lowering the performance of feature extractors. With these difficulties in mind, we use Genetic Programming (GP) techniques [18] to evolve a 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 to create high performance individuals 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. We then fine tune these explorations by identifying subtrees of well performing individuals that act as building blocks of success, where the GP system utilizes these subtrees to greater levels of success.

While the previously mentioned experiments use image data alone, the experiments in Chapter 4 explore sensor fusion. In these multi-sensor experiments we explore the suitability of using using image data alone, laser data alone, and the combination of image and laser data, by looking at the modes of error in each of these situations. We summarize these results with a discussion of the ways in which the addition of other sensory data can help or hurt in the evolution of successful individuals in our domain.

The work reported here is a direct extension of Martin's [23], and began as an attempt to test how well the results previously reported generalize across several environments. In the process, we sought to identify various techniques to help ensure the evolutionary success of a GP run. Such techniques form a set of GP best practices that are applicable to other problems in our domain, if not across others. We extended this work to use a laser range-finder, and then ran experiments using image and laser data combined.

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 discuss our multi-environment experiments and various techniques to help ensure high fitness levels. We describe our experimental setup, and then give the results of our experiments and offer some conclusions from this work. We then detail our experiments with image and laser data. We outline our experimental setup, and then give the results of our experiments. Finally, we offer some conclusions from this work and reflect on its value and possible future directions.

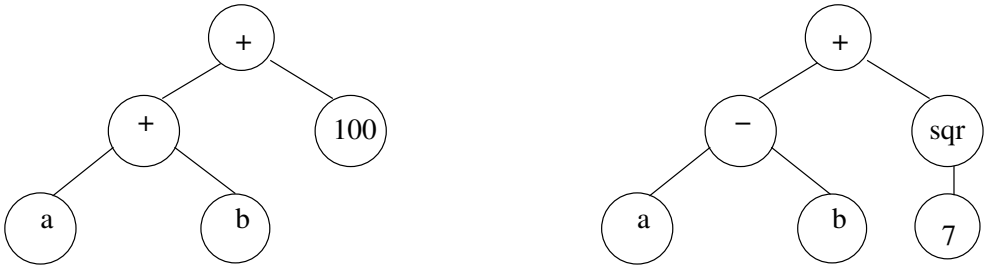
1.2 Genetic Programming

Genetic Programming (GP) is a programming technique inspired by genetics and evolution where solutions are evolved to solve a given problem. In order to demonstrate this technique, we use a very simple example where we try to learn a program that takes two integers as input, and produces as output, an integer as close to two hundred as possible (an easy problem for demonstration).

To start, the programmer needs to think about what kind of programming constructs are needed to solve the problem. Therefore, for this problem, we decide we need some arithmetic operators such as x, $/$, -, and $+$, and a squaring operator that takes in a parameter and squares it, called *sqr*. All of these programming constructs take parameters and are called *functions* in GP terminology. We also decide that we need some random constants such as: *2*, *7*, *100*, and two variables, a and b, to hold the two input numbers. The programming constructs that do not take parameters are called *terminals*. Together, the function and terminal sets are the *genes* that the GP system will use to construct programs (see figure 1.1).

Genes { Functions: x, /, −, +, sqr
Terminals: 2, 7, 100, a, b

Figure 1.1: Example function and terminal sets.



Individual 1: **(a + b) + 100** Individual 2: **(a - b)** + 7^2

Figure 1.2: Two example individuals.

Once the function and terminal sets are specified, GP randomly puts these genes together to form programmatically correct programs, or *individuals*. Two examples of such programs, and their parse tree representation, are shown in figure 1.2. GP creates thousands of these randomly constructed programs, thus, creating an initial population, or group of programs, as illustrated in figure 1.3. This size of the population is set by the programmer, and is a control parameter that we will discuss later.

Once an entire population is created, called generation zero, we need to evaluate each program in the population on how well they perform the given task, in this case, how close their output is to 200. For the two example programs shown in figure 1.2, if the both a and b are two, then program one and two evaluate to 104 and 49

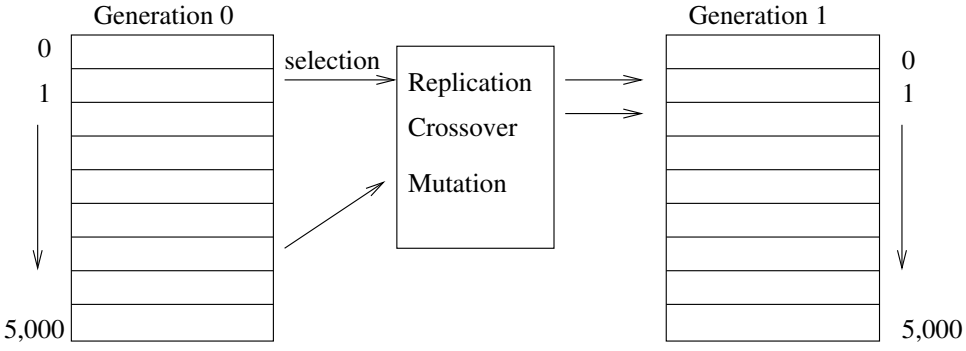


Figure 1.3: Population creation from generation to generation.

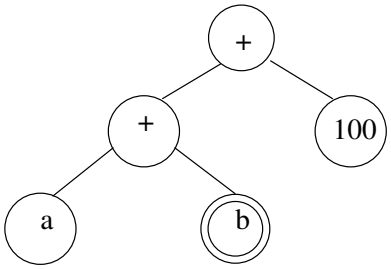
respectively. The evaluation mechanism, or fitness function, for this task is **| 200 - (program output) |**, thus, the two programs have fitness values of 96 and 151 respectively. GP is an optimization technique that minimizes or maximizes a function. In this problem, GP minimizes the fitness function so that a program that produces output closest to 200 is learned.

Once an entire population is evaluated, GP selects two individuals for the creation of a second population, or generation one. Many methods for selecting these two individuals exist, but they all are based on the notion that individuals that have better fitness values (either higher or lower depending on if it is a maximization or minimization problem) are more likely to be selected. With better performing programs more likely to be selected, the good traits that they possess (that helped them perform better), are likely to be passed on to the next generation.

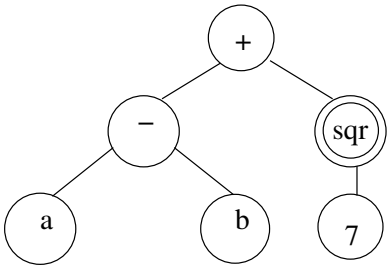
Once two individuals are selected, one of three genetic operators: replication, crossover, or mutation, is used to create two new programs to be inserted into the next generation (see figure 1.3). Replication is when a program is copied over verbatim into the next generation. Crossover, as shown in figure 1.4, works by selecting a random node in each of two programs and swapping the subtrees rooted at those nodes, then inserting both of these new individuals into the new population, or next generation. Mutation, as shown in figure 1.5, works by selecting a random node in one individual and randomly changing it to another member of the function and terminal set.

New individuals are created using these operators until a new population of programs of the same size as the previous population is created as shown in figure 1.3. At this point, the process starts all over again, with all of the programs in this new population being evaluated, with better performing individuals being more likely to be selected, and so on. This process repeats for a fixed number of generations, and typically, the individual with the best fitness over all inputs is designated as the result of the GP run. Throughout this procedure, creation, crossover, and mutation, are constrained so that resulting trees, or individuals, are viable programs.

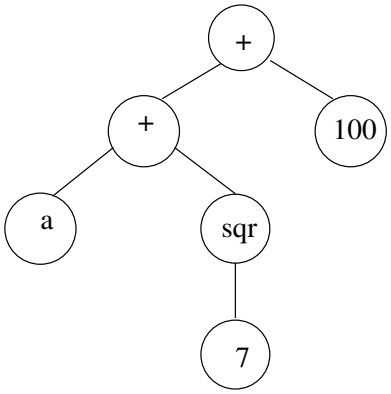
With this process, GP essentially performs a search through the space of possible programs, or solutions to this problem. It is more computationally efficient than an exhaustive search, but at the expense of the quality of the final solution. GP is not guaranteed to find the optimal solution, but often succeeds in finding a good one. GP is also able to overcome some of the local optima problems associated with local search techniques through genetic crossover and mutation, as these mechanisms can transplant a solution to an entirely different area of the search space.



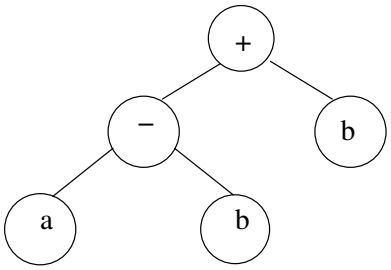
Individual 1: **(a + b) + 100**



Individual 2: **(a - b) + 7^2**

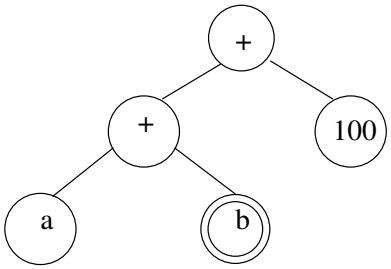


New individual 1: **(a + 7^2) + 100**

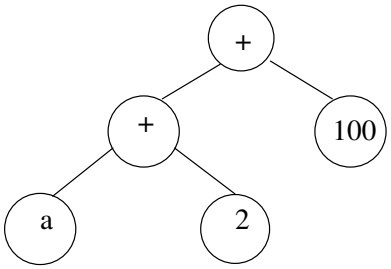


New individual 2: **(a - b) + b**

Figure 1.4: Result of crossover taking place in individual 1 at node *b* and in individual 2 at node *sqr*.



Individual 1: **(a + b) + 100**



New individual 1: **(a + 2) + 100**

Figure 1.5: Result of mutation taking place in individual 1 at node *b*. Node *2* replaces node *b*.

With regard to this search space, the programmer must decide on a number of control parameters before GP can start evolving programs. These decisions are important as these parameters affect how much of the solution space is covered during the search. The two main control parameters that need to be set are:

1. Population size: A larger population allows for a greater exploration of the solution space at each generation and increases the chance of evolving a good solution. In general, a more complex problem demands a larger population size.

2. Maximum number of generations: More generations give GP a better chance of evolving a solution, yet offer no guarantee that a solution will be found. In some cases, it might be best to start again with a different initial population.

With both of these parameters, it is often best for the population size and the number of generations to be as large as possible. However, GP is very computationally expensive, with complete evolutions often taking many hours if not days. For the computing resources at hand, the programmer needs to find a suitable trade-off between computation time and the quality of solutions generated, and set the control parameters accordingly.

Chapter 2

Motivations

2.1 Robot Obstacle Avoidance

Martin [23] 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 [13] with Polly the Robot.

Polly gave tours at the MIT AI lab, which has a textureless floor. Obstacles, or their boundaries, are therfore detected as areas of visual texture. Polly's vision algorithm starts at the bottom of the image and moves a rectangle, or window, vertically, until the algorithm detects significant texture. This texture is assumed to be an obstacle, and the further up the image this obstacle is found, the further away it is assumed to be. Navigation is then based on steering the robot towards the highest of the features identified. This Horswill inspired navigation system results in surprisingly robust navigation and is an important source of motivation for this work.

Figure 2.1 shows examples of the environment and the identified features in Martin's work. 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 used a tree-based representation, common to many GP approaches. The internal nodes of this representation corresponded to standard mathematical and control operations (such as looping). The terminals included five floating point registers, and a pre-defined set of primitive image operations, such as blurring and

Figure 2.1: Environment and detected features from Martin [23]. The white boxes indicate the evolved program's predictions for the lowest non-floor pixel in the column.

various edge detectors. In his early experiments, it was noticed that during the runs of many of the successful individuals, most of the time was spent performing a computation over a rectangular window which iterated over a column or row of the image. To remove the need to learn this control structure, it was explicitly added to the representation in the form of the *iterate* node. This primitive moved a rectangular window over an image, executing particular (learned) code at each location. Its (learned) arguments determined the size of the rectangle, the initial location, the direction and distance to iterate, and finally, a piece of code to execute at each location.

Experiments were performed with images taken from several locations in an office environment. In the initial set of experiments, the *iterate* nodes could operate either horizontally or vertically. The best individuals learned in these experiments performed only slightly better than programs that ignored the actual image data and just returned a fixed number. This happened despite a population size of 10,000 individuals running for 51 generations. Individuals were limited to a total size of 6000 nodes.

However, when a poorly performing hand-written program was used to seed the population, much more successful individuals were learned. The best such individual achieved an average error of only 2.4 pixels, and identified 60% of the test cases within 2 pixels of the human-provided ground truth position. Martin reports subjectively that the vast majority of fitness cases were handled more than well enough for obstacle avoidance, and the errors did not follow any particular pattern, suggesting a simple filter would eliminate most of them.

While many aspects of the seed individual were modified by the evolutionary computation, others were not. In particular, the successful evolved algorithms all iterated vertically, from the bottom of the image to the top or vice versa, just as the seed did. Therefore, in a second set of experiments, horizontal iteration was

eliminated and the iterate node was simplified to take only three arguments: the window size, the horizontal location in which to iterate, and the code to execute at every step. The result producing branch, which had simply returned the result of a single iteration branch, was also eliminated.

These new experiments produced individuals which routinely achieved a fitness of greater than 85% [23, page 117]. They did this despite image features such as burned out lights and other lighting effects that caused the carpet's average greyscale intensity to vary between values of 0 and 140 (out of 255). They also coped with large gradients caused by imaging artifacts, moiré patterns of image noise, and the shadow of the robot. The results from these experiments were more than good enough for navigation. Once again the vast majority of columns were interpreted correctly and with one exception errors were transient. The one exception was a particular stripe of red carpet in one corridor, which was uniformly considered an obstacle, at least when near the robot. This suggests that a richer representation of the image, perhaps using color rather than greyscale, might be more robust.

This work is described in more detail by Martin [23], and forms 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 at using GP for learning programs for more general feature detection problems.

2.2 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. Also similar is work by Johnson [16], which uses GP to learn visual routines [26] that were used to determine human actions in the ALIVE virtual environment [22]. The learned routines ran over silhouette images of a human, and detected things like hand position. The system uses a specialized LISP-like language, similar to that used by Martin, and is capable of learning complex programs (on the order of hundreds of expressions).

Koza [19] used GP to learn programs for a simple optical character recognition task. More in-depth research in the same domain was carried out by Andre [2]. Teller and Veloso used GP to learn programs that classify images in the PADO system [25].

Harvey, Husbands and Cliff [11] use GP to evolve the control and morphology of a very 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.

Graae, Nordin and Nordahl [10] evolve a program to calculate disparity from a pair of greyscale stereo images, using a sliding window representation, similar to that used by Martin.

Lutton and Martinez [21] 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 [14]. 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.

Poli [24] uses a parse-tree based representation to evolve image filters. There are three terminals, all square blurring convolutions (2x3, 7x7 and 15x15), and six basic functions (add, subtract, multiply, divide, max and min). The learned programs were used to perform segmentation in medical images. They claim that the GP-based approach was more successful than neural network methods and "many other techniques reported in the image-analysis literature".

Dumoulin *et al.* [7] use genetic algorithms to design image convolution operators that are implementable in reconfigurable hardware (FPGAs). Their program representation involves very simple bitwise operations and control flow, as one might expect when using FPGAs.

Not using genetic programming, but similar in spirit to the work described here, Draper, Bins and Baek [6] 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 [15].

Chapter 3

Cross Environment and General Performance Enhancement

3.1 Experiments

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

3.1.1 Environments

We gathered data from four different indoor environments, examples of which are shown in figure 3.1. The details of these environments are as follows:

A1 Jolley Hall - Office/corridor environment with sharp shadows from strong sunlight coming through open doors an 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.

B Eliot Hall - 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 Olin Library - 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.

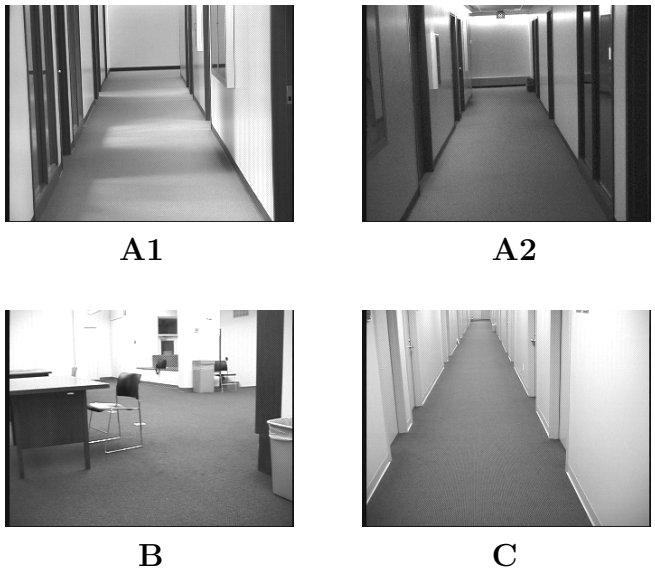


Figure 3.1: The four test environments. The checkered pattern evident in some images is a printing artifact and is not present in the actual environments.

We took 351 frames of video (320 x 240 pixels) while moving through each of these environments (frame0.pgm - frame350.pgm), and used every 5th frame starting at frame zero to construct the training sets, and every 5th frame starting at frame one to construct the test 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 (70 frames * 6 columns) fitness cases. If the predicted positions are within ten 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.

3.1.2 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 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, as described in Martin's work in section 2.1, only vertical iteration was used with the iterate node taking

three (learned) arguments: the window size (area of the image the program has access to), horizontal location, and the sub-program to execute at every step. 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 3.1.1.

Martin experienced little improvement after the 50th generation [23]. For this reason, we limit ourselves to 50 generations of evolution, with a population size of 4,000. 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. An individual's size is checked during crossover such that if it exceeds 1000 nodes, the individual is not used, and another attempt at crossover is made. 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.

3.2 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 mediocre fitness. We discuss this further below.

3.2.1 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 3.2. Individuals evolved for a particular environment do better there than in any of the others. Additionally, higher fitness was observed

Table 3.1: List of available functions and terminals to cross environmental and general performance experiments.

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 (<= 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 3.2: Fitness of best individuals after 50 generations of evolution. Best fitness for each test set is underlined.

	Test Data Set				
Train	**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%

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 3.1). 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 training set **B** is almost enough on its own to produce a robust algorithm that works across environments.

Indeed, the fitness of programs trained 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. During the cross environmental experiments, we saw an individual produced by a difficult, unstructured, enironment perform well across all environments. This difficulty in the training data forces the resulting program to evolve controls that do not rely on the environment's structure, and consequently, creates a program that works well in multiple environments. We also noted that diverse training data, like the combined data set, produces individuals that generalize well across environments. Such diversity in training data better prepares the resulting programs to deal with a variety of situations successfully.

3.2.2 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

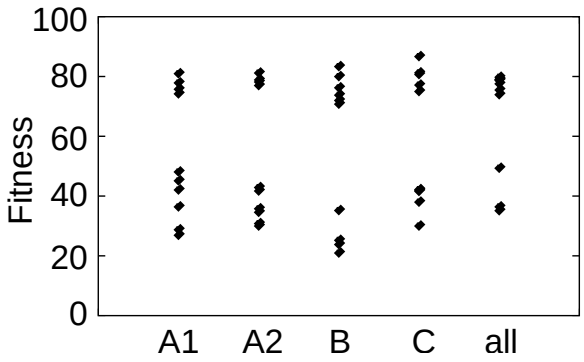


Figure 3.2: The bimodal distribution of the best individual fitnesses after 50 generations.

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 3.2, 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.

In order to test this hypothesis we chose to examine the **A1** data set. Since the learned programs are very large, we chose the single best run from a particular experiment, and looked at how it changed with each generation. Most increases in fitness are small, except between generations 16 and 17, where there was a 9% increase, and between generations 18 and 19, where there was an increase of approximately 7.5%. While these fitness jumps are characteristic of evolutionary computation, they seem to indicate that some important part of the feature detection program has been discovered.

Analyzing the evolved individuals proved to be extremely difficult. Initially, instead of identifying any particular key part of the individual, we re-ran the experiment, with the best individual from generation 19 in the initial population. All other initial individuals were created randomly. The results of the best individual from

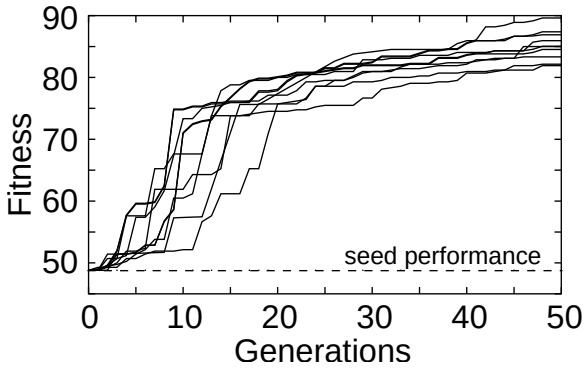


Figure 3.3: Ten best individuals from a seeded run in **A1**.

ten runs of the experiment are shown in figure 3.3. All parameters are the same as described above, and we ran each experiment for 50 generations of evolution.

In each of the ten experiments, the best individual performed well, with fitness between 81.9% and 89.6%. In all ten runs, the seed (dashed line in figure 3.3) was modified and improved upon considerably. Even the *worst* fitness here, is slightly better than the *best* fitness in the original set of experiments for this environment (see table 3.2).

Seeding in this manner has three advantages over using a hand-written seed. First, we do not need any knowledge of the problem domain or of its specific programming needs. Secondly, we do not need to know how to create a tree by hand. And thirdly, an evolved seed will have redundant code. Thus, it is not as brittle as hand-written, compact code when subjected to GP operations. Because of the redundancy of the evolved seed, non-catastrophic changes *can* be made to it, and because of its size, there are many opportunities for crossover to select a working sub-program. We believe that these reasons make the use of a less finely-tuned seed (one created from evolution) a better option, at least in the domain in which we performed these experiments, especially when hand-coding an appropriate individual is too difficult or is unsuccessful.

We then sought to identify particular features, or function combinations, within individuals, that are key to high fitness. Again, we examined the single best run from data 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 persistance 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

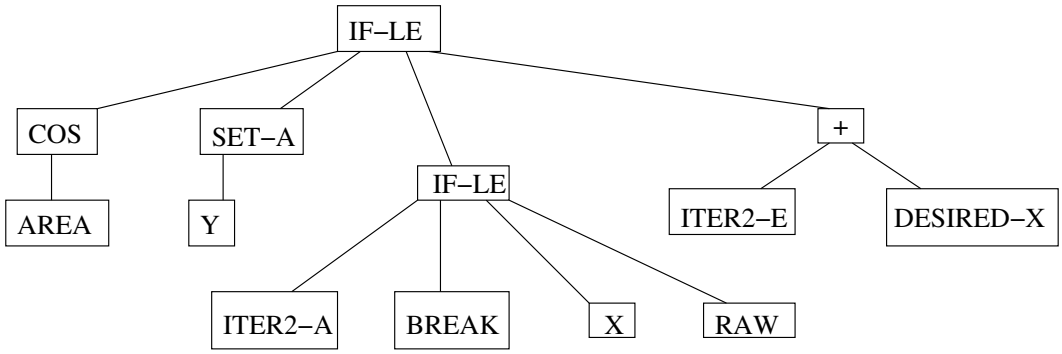


Figure 3.4: Subtree 1.

Table 3.3: 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.

	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%	

start. We identified two key subtrees in this manner and executed ten runs with the first subtree and ten runs without. The two subtrees are shown in figure 3.4 and figure 3.5. The results of these experiments are shown in table 3.3.

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 (80% confidence level).

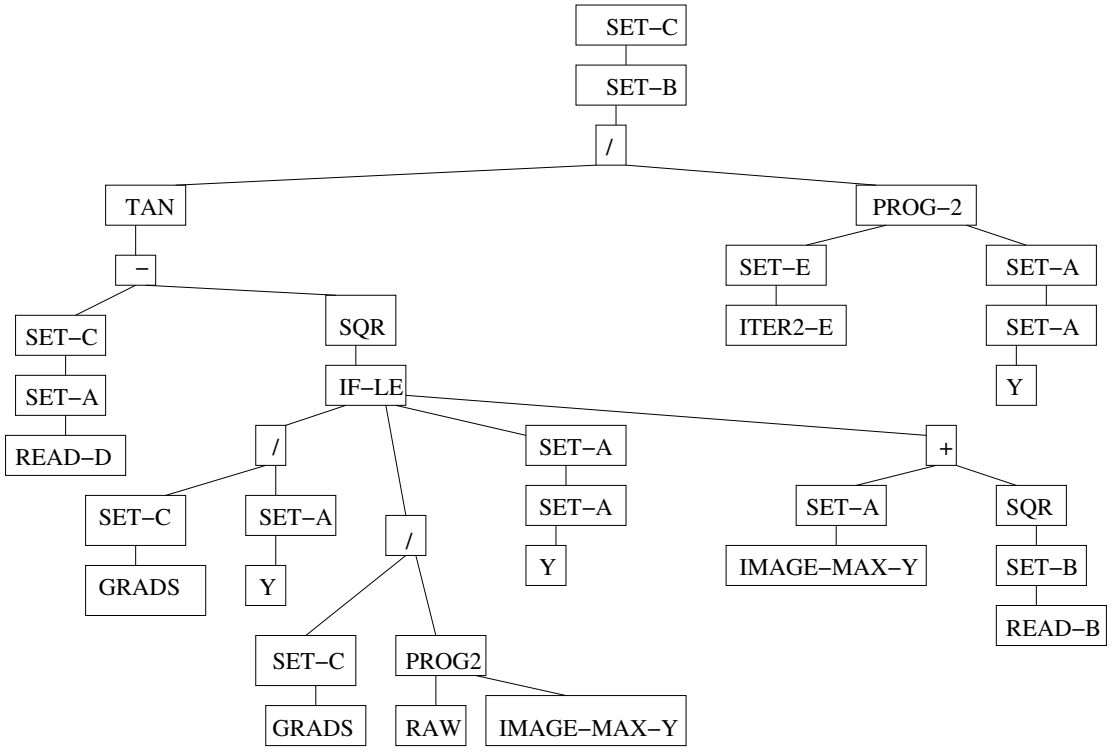


Figure 3.5: Subtree 2.

Chapter 4

Multiple Sensor Fusion

4.1 Experiments

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

4.1.1 Data Collection

For these experiments, we captured data from the Lewis the robot as it moved around the Media and Machines lab at Washington University. Lewis is an iRobot B21r mobile robot with its video camera mounted on a Directed Perception pan/tilt unit. The robot is a red cylinder that is over four feet tall and approximately two feet in diameter. We used one video camera, a Sony DFW-VL500 IEEE 1994 (FireWire), and the laser range-finder, which returns 180 radial distance measurements, covering the front 180° of the robot, at a height of approximately 40cm above the floor. In order to limit the laser range-finder data to scan values that correspond to image data, we eliminated laser scan values outside the field-of-view of the camera. We eliminated these scan values by setting up two obstacles high enough to be detected by the laser range-finder, at the left and right edges of the camera's field of view as shown in figure 4.1. During this process, we erred on the side of caution so as not to remove any laser scan values that were within the field of view.

With the procedure above, the first 63 and last 63 laser scan values were removed, leaving 54 laser scan values that fit within the robot's field of vision. With Martin and Horswill as our main motivations for these experiments, the structure of the vision and navigation algorithms centers around the notion of exploring the

Figure 4.1: Two obstacles at the left and right edges of the image are used to eliminate laser data that is outside the robot's field of view.

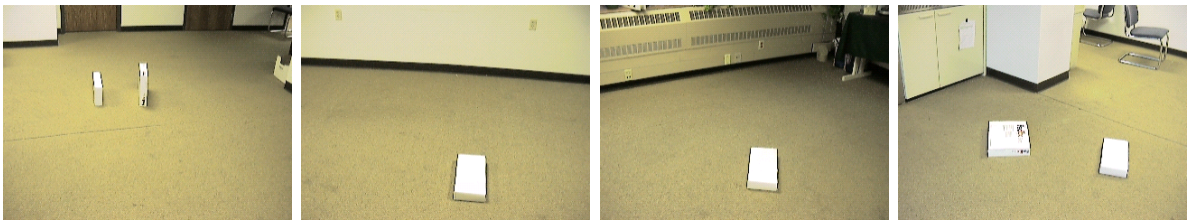


Figure 4.2: Washington University's Media and Machines laboratory where the training and test data were collected for the multiple sensor experiments.

world around the robot in columns. In keeping with this column centered approach of the vision algorithm, we hoped to provide the GP system with meaningful laser data that represents the robot's perception of the world over each column in the image. In order to accomplish this goal, we divided the available 54 laser scan values, into nine equal columns (as with the image data), such that the GP system can utilize, average, and process six laser scan values for each column. Consequently, individual laser scan values do not have to correspond exactly to specific points in the image. This process afforded us more time to concentrate on evolving good individuals, and we suspect that we did not suffer any decrease in individual fitness as our resulting individuals attain very high fitness levels.

4.1.2 Environment

The Media and Machines laboratory, as shown in figure 4.2, is a typical office/corridor environment. Data were collected at night, with the presence or lack of overhead lighting producing the variations in light intensity. The walls are white with dark molding at the bottom. The carpet is lighter than the molding, but darker than the walls.

We collected two data sets, each consisting of 80 frames of video (40 training and 40 for test), and recorded the corresponding laser range-finder data to a file while moving through this environment. One data set, **low**, consists of objects, the two white boxes in the images above, that are too low for the laser range-finder to detect. In order to establish ground truth information for each of these frames, we had to capture a frame with the boxes situated below the laser range-finder, and then, capture the laser data with the boxes rotated upwards such that they are detectable by the laser range-finder. The boxes are rotated such that their longest sides run perpendicular to the floor, they are located in the same place, and show the same area to the robot. Any error in this repositioning is nullified due to the leeway afforded by the column approach. The second data set, **visible**, consists of the same white boxes in the images above, arranged so that they *are* visible to the laser range-finder.

After the data set with the boxes rotated upward is collected for data set **low**, and after data set **visible** is collected, a human then specifies which of the nine columns contain "obstacles of interest", or boxes. To evaluate fitness, we compare this ground truth information of whether a column contains a box or not for all of the frames in a data set with the GP generated predictions. With nine columns per frame and forty frames, we have a potential for 360 fitness cases. With the boxes being present, on average, in two of the nine columns (22.22%), it became clear we had to direct the GP system to evaluate certain fitness cases. If we made no adjustment, a prediction of no obstacles of interest for all fitness cases would yield a relatively high fitness level of 77.78%, and the GP system would have little incentive towards finding a solution that actually detects boxes.

Therefore, we wanted to attain a more equal look at fitness cases where an obstacle of interest is present, and where one is not. We ensured an equal look by constraining the system such that the number of fitness cases where no obstacle is present equals the number of cases where an obstacle *is* present. In these experiments, 85 columns contained an obstacle of interest and 85 did not, for a total of 170 fitness cases. For a given individual, total fitness is the number of correct predictions over the total number of fitness cases.

4.2 Laser Range-finder Experiments

As the description of the ground truth and data collection process suggests, we want to be able to identify if a given column in a laser scan contains a box. For these

Table 4.1: List of available functions and terminals to box detection experiments using laser only.

arithmetic	*, +, /, -, 0, 1, and random constants, threshold function that returns 1 if the first parameter is <= the second parameter, 0 otherwise
flow control	<= operator
laser statistics	average scan value over a column, difference of current column average and previous column average, difference of current column average and next column average, minimum scan value over a column, maximum scan value over a column

experiments, we specified nine evenly-spaced columns to divide up the 54 laser scan values. Fitness was evaluated as described above in section 4.1.2.

4.2.1 Experimental Setup

In these laser-only experiments we limit ourselves to 50 generations of evolution, with a population size of 4,000. Individuals are chosen using tournament selection with a size of seven. Genetic operators include crossover (90% rate), and reproduction at a rate of 10%. In order to prevent code bloat, individuals created through crossover are limited in size to a tree depth of 17. We chose a tree depth of 17, because 17 is the default tree depth for Open BEAGLE, the GP framework we chose for this work, and it was a depth that worked well for many fellow BEAGLE users. See appendix A for more details about Open BEAGLE. Mutation is not used, since, empirically, crossover seems to introduce enough disruption onto the population. The functions and terminals available to the laser-only runs consist of arithmetic, flow control, and laser statistics, as shown in table 4.1.

4.2.2 Experimental Results

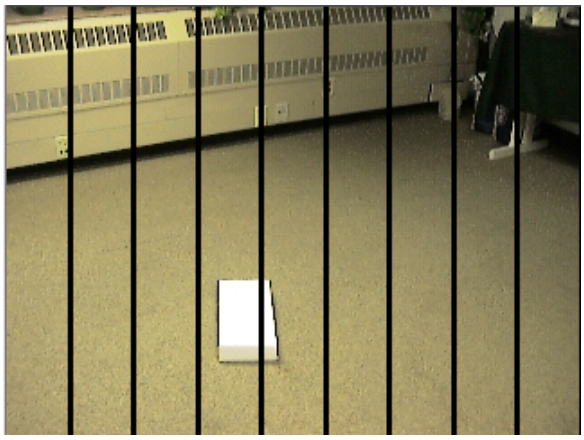
We begin this section by discussing the performance of the GP system in its attempt to detect boxes using the laser range-finder alone. We then discuss the behavior of these evolved individuals below.

4.2.3 Box Detection With a Laser Range-finder

In these experiments, we first trained the system on data set **low** where none of the boxes can be detected by the laser range-finder. Therefore, we are not evolving a box detection algorithm, but rather challenging the GP system to find and to exploit patterns in the training data to produce individuals of high performance. Indeed, the resulting individuals from these experiments had moderate "success" in detecting these boxes, or rather in exploiting regularities in the environment.

The best performing individual from the 30 GP runs was 74.45%, with all 30 runs averaging 71.55% with a variance of 3.45%. With these results, it is clear that the GP system made use of regularities in the environment. As with many GP evolved programs, it is very difficult to ascertain the functionality of the program. The best performing individual in these experiments contained 687 nodes and had a tree depth of 16. Yet, after analyzing the predictions for each column in the data set, it appears that the individual made use of certain patterns in the environment and box placement. It appeared to mark columns as interesting that had an average scan value less than 600 cm and that had small differences (40 cm) with the average scan values to the left and right of the current column. Examples of this behavior are shown in figure 4.3 and figure 4.4. This sort of behavior is representative of the type of adaptive behavior that programs evolved through GP display, where individuals make use of the information they are provided and adapt to the features of the environment. Such patterns in the environment may or may not have been apparent to a human programmer, and thus, GP shows its value as a pattern identifier, even in the most unlikely of circumstances. However, the patterns identified here are highly specific to the environment, and while the resulting fitness was achieved on test data, they would surely fail in other environments, resulting in poor performance, and poor generalization.

We then trained the system on data set **visible**, where all the boxes were visible to the laser range-finder. The best performing individual from 30 GP runs on this data set was 97.06%, with all 30 runs averaging 95.06% with a variance of 3.09%. These results show that laser alone is enough to detect boxes in this environment, with misclassifications occuring where the box extended slightly into a column that was not marked as containing a box. It is likely that much of the success experienced here is due to regularities in the environment as demonstrated in the results of training where none of the boxes are visible. The combination of patterns in the environment and detectable obstacles made this a trivial task for the laser range-finder alone. Again,



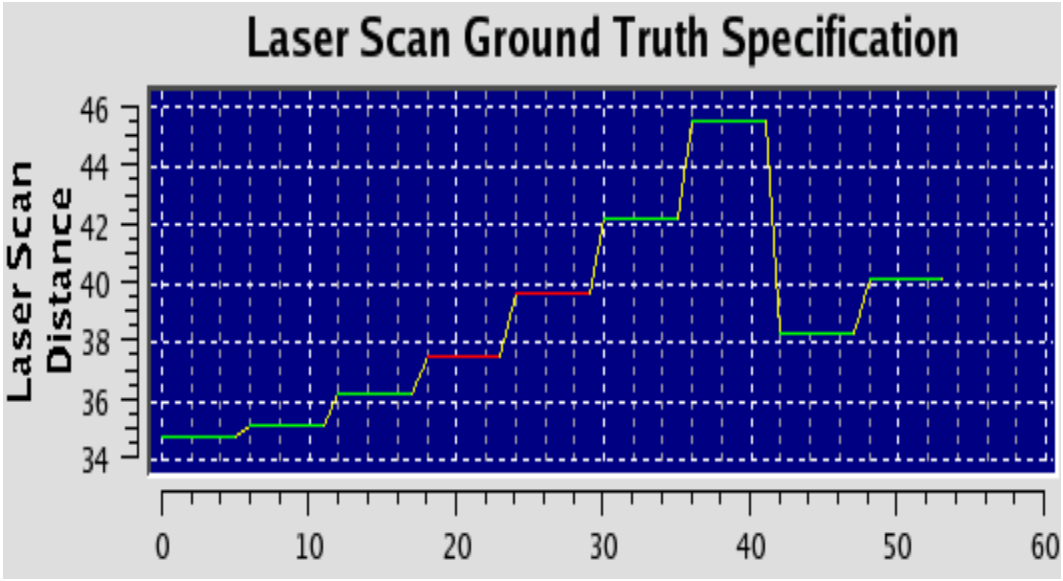
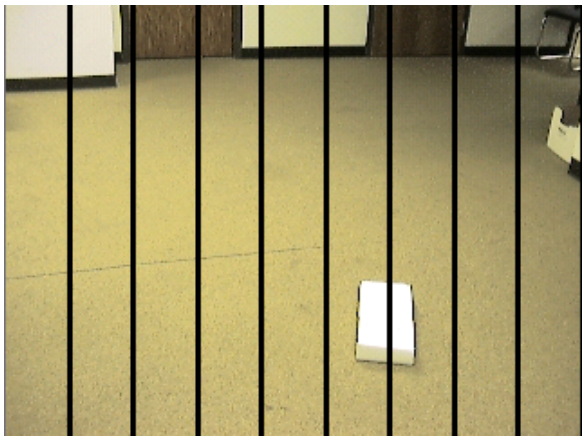


Figure 4.3: An image divided up into nine equal columns, along with its corresponding average laser scan values for each column. Red means the system predicted that an "obstacle of interest" is present, and green means that no prediction was made. To get distance in cm, multiply y-axis by 10.



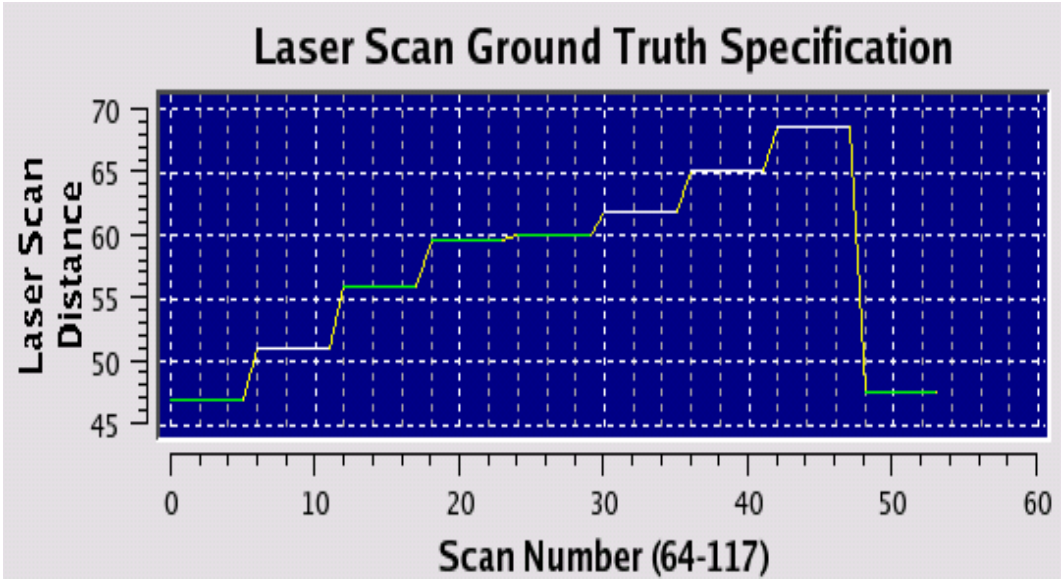


Figure 4.4: White means the system predicted that an "obstacle of interest" is *not* present, and green means that no prediction was made. To get distance in cm, multiply y-axis by 10.

Table 4.2: List of available functions and terminals in the main tree in the box detection experiments using image only.

arithmetic	*, +, /, -, 0, 1, and random constants, threshold function that returns 1 if the first parameter is <= the second parameter, 0 otherwise
flow control	<= operator, ADF

the individuals evolved here are likely to achieve lesser fitness levels when tested in other environments due to over-fitting.

4.3 Vision Based Experiments

In these image-only experiments, we want to be able to identify if a given column in an image contains a box. For these experiments, we specified nine evenly-spaced columns in each image with fitness evaluated as described above in section 4.1.2.

4.3.1 Experimental Setup

The number of generations, population size, and genetic operators, are the same as those described in section 4.2.1. The list of functions and terminals (see table 4.2) available to the image-only experiments is similar to those available to the laser-only experiments except for the addition of an automatically defined function (ADF) and the removal of the laser statistics. The ADF is composed of the same functions and terminals and returns the same value (register a) as the cross environment experiments as shown in table 3.1. ADFs are subtrees that are used as functions in the main tree[18]. They evolve separately from the main tree and use separate function and terminal sets to create programmatically correct trees.

4.3.2 Experimental Results

We begin this section by discussing the performance of the GP system in detecting boxes with the use of the image data only. We discuss the nature and details of these evolved individuals below.

4.3.3 Vision Based Box Detection

In the vision-only experiments, where all boxes were visible to the camera, the best performing individual did quite well by only misclassifying five of 170 fitness cases for a fitness of 97.06%. After 30 runs, individuals averaged a fitness of 87.73% with a variance of 34.47%. Here, the best resulting individual contained 420 nodes and had a tree depth of 16. Misclassifications were caused by other lightly colored objects such as a silver chair leg, or lightly colored table leg. Overall, however, the best performing individuals from these experiments were extremely successful in identifying boxes in a variety of conditions.

4.4 Sensor Fusion Experiments

In these experiments, we combine image and laser range-finder data. Here, we specified nine evenly-spaced columns in both, the images and the laser scans, with fitness evaluated as described above in section 4.1.2.

4.4.1 Experimental Setup

Here, we use the same setup, with regard to number of generations, population size, and genetic operators, as the image-only and laser-only experiments described previously. The list of functions and terminals available to these sensory fusion experiments are a combination of those available to the image-only and laser-only experiments. The genes available to the main tree are the same as those available to the laser only experiments, as shown in table 4.1, except for the addition of an automatically defined function (ADF) that envokes the image specific code. The ADF returns the value in register a and is composed of the same genes as the ADF in the image-only experiments as shown in table 3.1.

4.4.2 Experimental Results

We begin this section by discussing the performance of the GP system in detecting boxes with the use of the image and laser range-finder data. We discuss the nature and details of the evolved individuals below. We also give a more detailed comparison of the resulting individuals from each of these experiments with image data only, laser range-finder data only, and data from the two sensors combined.

4.4.3 Box Detection With Sensor Fusion

The first set of experiments used data set **low** where all boxes are undetectable by the laser range-finder. The best performing individual from these experiments classified 168 of 170 fitness cases correctly for a fitness of 98.82%, with 30 runs averaging 90.82% with a variance of 54.37%. Here, the best resulting individual contained 508 nodes and had a tree depth of 16. The predictions for each column in the data set were correct for almost every fitness case. Misclassifications occured because of the recurring issue of when a box extended slightly into a column that was not marked as containing a box. Overall, however, the best performing individuals from these experiments were extremely successful in identifying boxes in a variety of conditions.

However, with the laser range-finder unable to detect the boxes, we cannot say that high performance individuals in this experimental setup made use of both sensory modalities, but rather, they leveraged useful image data along with regularities in the environment. Our laser range-finder only experiments show how without any useful laser data, the GP system successfully found and exploited patterns in the environment for better performance. Therefore, we can assume that the increased performance noted here, with the addition of *blind* laser data, is a result of patterns in the data set and not successful sensory integration.

The second set of experiments that attempted to fuse vision and laser data used data set **visible**, where all boxes were detectable by the laser range-finder. The best performing individual from these experiments classified 164 of 170 fitness cases correctly for a fitness of 96.47%, with 30 runs averaging 94.23% with a variance of 1.62%. Here, the best resulting individual contained 319 nodes and had a tree depth of 16. Misclassifications occured where a box extended slightly into a column that was not marked as containing a box. Overall, however, the best performing individuals from these experiments were extremely successful in identifying boxes in a variety of conditions.

The results from all of these multiple sensor experiments are summed up and compared in table 4.3 and table 4.4. The results from the second set of experiments, as shown in table 4.4, where the boxes are visible to the laser range-finder, show that either the laser range-finder alone, or the camera alone is sufficient in identifying boxes. The laser range-finder runs and image-only runs had means of 95% and 88% respectively, and best runs both equaling 97.06%. In fact, all runs with the laser range-finder recorded fitness levels above 90%, with a small variance of 3.09%. The results from the image-only runs show that runs with image data alone are very good

Table 4.3: Fitness with access to different sensor data where boxes are too low for the laser range-finder.. The t-statistic is for the t-test of the hypothesis that for each pair above, the fitness of the first experiments (from left to right) is greater than the fitness of the second group of experiments. Confidence is the corresponding confidence level at which this hypothesis holds.

	Image vs Laser		Both vs Laser		Both vs Image	
Mean	87.73%	71.55%	90.82%	71.55%	90.82%	87.73%
Variance	34.47%	3.45%	54.37%	3.45%	54.37%	34.47%
t-statistic	14.36		13.86		1.73	
confidence	99.99%		99.99%		95.5%	

Table 4.4: Fitness with access to different sensor data where boxes are detectable by the laser range-finder. The t-statistic is for the t-test of the hypothesis that for each pair above, the fitness of the first experiments (from left to right) is greater than the fitness of the second group of experiments. Confidence is the corresponding confidence level at which this hypothesis holds.

	Laser vs Image		Laser vs Both		Both vs Image	
Mean	95.06%	87.73%	95.06%	94.23%	94.23%	87.73%
Variance	3.09%	34.47%	3.09%	1.62%	1.62%	34.47%
t-statistic	5.9		1.3		5.9	
confidence	99.99%		89%		99.99%	

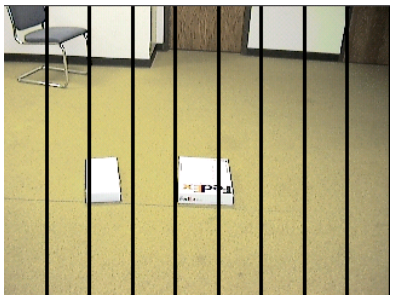


Figure 4.5: Image-only failure mode with silver chair leg being misclassified as a box.

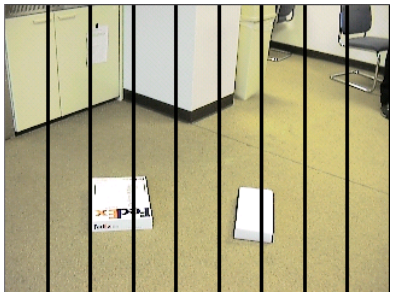


Figure 4.6: Image-only failure mode with wall protrusion located between two boxes being misclassified as a box.

at identifying boxes, but suffer from a lot of volatility. While all runs were over 70% with the image data alone, they varied from 70% to 97%. Another important fact is that programs evolved with image data only took an average of 22 hours to complete, while runs with laser data only averaged 30 minutes.

The performance of the laser-only and image-only runs, where the boxes are visible, are predictable, in that the alogrithms fail in predictable ways. After looking at the best run from the image-only experiments (97.06% fitness), all but one of the failures were caused by other light colored objects such as a silver chair leg, or lightly colored table leg. Figure 4.5 shows one such case where the second column from the left, containing the chair leg, was misclassified as having a box. The only other failure occured when a column containing a protruding wall structure was located in between two boxes, as shown in figure 4.6. We assume that, in this case, the combination of the unique wall protrusion, the closeness of the protrusion to the two boxes and the robot, and the stark contrast between the dark molding and the white walls, made this a tough case for the alogrithm. The alogiritm did classify all other columns with this contrasting molding and wall pattern correctly, however.

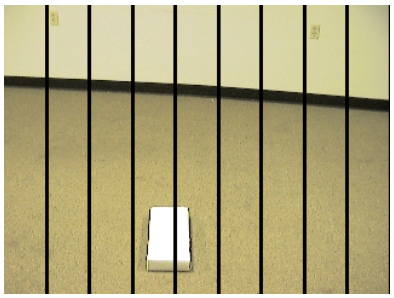


Figure 4.7: Laser-only failure mode with column overlap causing a box classification where one was not specified.

After examining the best run from the laser-only runs (97.06%), we noted that all failures, or misclassifications, were due to boxes that extended into columns that were not marked as containing boxes. Figure 4.7 shows an example of this column overlap. This overlap is a problem in laser-only runs, as opposed to image-only runs, because all laser statistics are based on an average value for a given column. Therefore, small overlaps that did not warrant a classification as containing a box, make significant differences in the column average nonetheless. This sort of error is avoided in image-only runs because the image statistics are centered around a specific "x" value within the column, and are not forced to consider all of the surrounding information in that column. This laser-specific failure mode suggests that columns containing any part of a box should be marked as containing a box, as opposed to limiting box classifications to boxes that extend over half or more of the column. With only five miclassifications due to this overlap, however, and with adjacent columns being marked as containing an obstacle, the best run individual is more than adequate for the task of box detection, especially when used for obstacle avoidance.

After close inspection of the failure modes of the best run individual from the image and laser data combined runs (96.47%), we found that the misclassifications in this case, were a combination of the errors with laser data only and the errors with image data only. The best run individual failed when a column contained a wall protrusion located between two boxes as in figure 4.6 from the image-only experiments. This combined data algorithm had no trouble, however, with chair or table legs. The algorithm failed again when boxes extended into columns that were not marked as obstacles, but not in all of the cases that we saw in the laser-only runs. This behavior suggests that the algorithm possibly made use of image data to help prevent it from classifying the column incorrectly, but was not successful in combining the sensors in all cases. Indeed the combination of the laser and image data did

not produce greater success, but rather had difficulty combining the two sensors to form better solutions than image or laser data alone. The combined sensor runs were far less volatile (1.62%) than the image-only runs (34.47%), as the presence of laser data proved to be a steadying influence. However, the combined data runs were less volatile than the laser-only runs, suggesting that the combination of the two sensors increase the liklihood that a high fitness (over 90%) individual will be evolved.

Chapter 5

Conclusions and Reflections

5.1 Conclusions

In this thesis, we have presented the results of using GP to learn feature detectors with multiple sensors. In the vision domain, we discussed the results of experiments across several environments, and analyzed a bi-modal learning behavior, which resulted in the final performance either being very good or very bad.

We explored various techniques to help ensure that individuals achieve high fitness levels, and although we dealt with a particular set of experiments in a particular domain, we believe that this work 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.

In addition, we have demonstrated that evolved subtrees, or terminal and function clusters, can increase GP performance significantly. The use of evolved subtrees also provides the opportunity to test 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.

The research done here also supports the idea of having difficult and diverse training data so that the evolved individuals generalize well to other applications.

We also investigated the use of multiple sensors in a series of experiments where either, just image data, just laser data, or a combination of the two are used to evolve a box detection algorithm. We showed that in the absence of useful data, or in the presence of misleading data, GP will look for patterns in the environment in order to

achieve higher fitness levels. The GP system also successfully found these patterns when they seemed not to exist to a human. Through our experiments where boxes are too low for the laser range-finder, we highlight the danger in picking useless, or misleading, functions and terminals (the laser statistics), as doing so leads to over-fitting to the environment.

We found that when all boxes are visible to the laser range-finder, image data alone, or laser data alone, are sufficient for box detection. Laser-only runs were much faster and much less volatile than the image-only runs. Combining image and laser data did not lead to higher fitness values, but did create less volatility and a greater probability that performance would be very good (over 90%).

Throughout all of these experiments, we found that the success of GP increases tremendously, when using camera and laser range-finder data, when the user has some knowledge of the problem and of the pieces that make up the solution. In this manner, GP becomes an effective directed search, with its direction provided by the choice of operators, functions, and seeds. We believe that many of the principles learned here can be applied in a variety of problem domains, especially in the areas of general feature detection in many environments.

5.2 Reflections and Future Work

Overall, we found the domain of general feature detection using multiple sensors to be a very useful platform for exploring GP. We learned a great deal about training data characteristics, and other techniques such as seeding, that are useful in ensuring higher fitness values in our domain and others. We have just scratched the surface with regard to combining multiple sensors and the type of problems GP can handle successfully. While the work here is a step in the right direction, this domain deserves a more comprehensive study. We believe this work provides a complete, easy-to-use framework that covers: data collection, off-line training, testing, and online validation on the robot.

In the event that we had more time, we would first combine the two data sets that we have already collected, **low** and **visible**, and train the GP system with this combined data set. Instead of treating the boxes that are too low for the laser range-finder as boxes, we would classify them as pieces of white paper on the floor. As Martin and we have seen, sharp contrasting carpet patterns and shadows often fool a vision system that associates light intensity differences with obstacles. In our

experiments, the vision system is detecting the intensity differences between the white boxes and the darker carpet. By treating boxes that are too low for the laser range-finder as pieces of white paper, we are challenging the GP system to come up with another means of detecting boxes. It is likely that with this data set, the two sensors will disagree, with the vision system classifiying a piece of paper as a box, and the laser range-finder not detecting this "box". Such sensory disagreement would perhaps force the GP system to use the two sensors in combination in a more comprehensive manner than we have seen thus far. The hope is that the GP system would use the laser data to correctly classify a column containing a piece of paper as obstacle free, and columns containing boxes as obstacles to avoid.

Additionally, with the goal of a comprehensive robot demo, we propose the following experiments. First, collect three more data sets:

1) A typical office/corridor environment free of boxes or other obstacles except walls

2) Same as data set 1, with the addition of an equal mix of white boxes and pieces of white paper on the floor

3) Same as data set 1, with the addition of tables along the walls

Once these data are collected, train the GP system on each of the three data sets, using just image data, just laser data, and the two sensors combined.

Instead of outputing whether or not an "obstacle of interest" is present in each column, output the lowest non-floor pixel in each column. Our hypotheses for these experiments are:

1) Image data alone are enough to successfully identify walls and boxes

2) Laser data alone are enough to find boxes that the laser range-finder can detect

3) Image data alone are not enough when high contrast floor features are present such as white pieces of paper

4) The use of laser and image data combined prevent errors with the sheets of paper

5) The combination of laser and image data will allow the detection of the correct position of tables, where using each sensor alone will not

With the current ability to transform the BEAGLE XML output to C code, run this code on the robot in order to successfully detect boxes, walls and tables. Then,

feed this obstacle detection output to a Polly navigation algorithm to show the robot avoiding the previously mentioned obstacles, while running over pieces of paper.

Appendix A

Open BEAGLE

Framework Open BEAGLE is a C++ Evolutionary Computation framework that supports tree-based Genetic Programming, bit-string and real-valued genetic algorithms and multiple evolutionary stategies. We selected this framework for our multiple sensor experiments because it is well-designed and stable. BEAGLE's excellent use of Object-Oriented programming techniques, advanced debugging and tracking features such as a log file, and an XML-based file format for configuration and representation of individuals are features that provide ease-of-use, robustness, versatility, and portability. Its current support of advanced techniques such as co-evolution and evolutionary strategy, as well as features currently in development, such as support for distributed programming are excellent candidates for future research and additional reasons why we selected BEAGLE as our GP framework.

Modifications While BEAGLE provided a versatile and robust framework for our work, we had to make some modifications to the code in order to support the structure of our individuals. The individuals in our work have a structure where an iterate node is at the root, and takes in three arguments: the rectangle size, the column to iterate over, and the program to execute at each step as shown in figure A.1. The built-in creation methods in BEAGLE do not support this structure where two of the children of the root node are constrained to be terminals, and the third child, the program to execute at each step, is constrained to have a minimum depth of 2, or in other words, to be a function. The programmer sets a parameter to the creation methods that specifies the minimum depth of the trees created. In order to promote well developed and interesting trees, we set the minimum depth to 5, as did Martin in his work. The creation methods in BEAGLE only select functions when the minimum tree depth

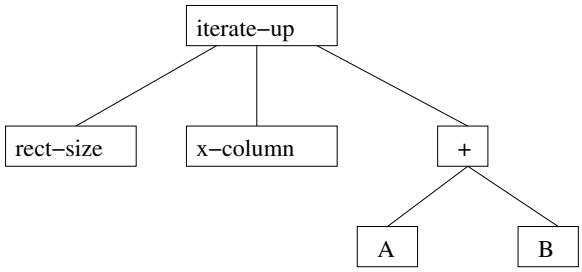


Figure A.1: Example individual with Martin's structure.

is not reached, consequently, failing when it tries to add functions as the first two children to the root node. In order to prevent this failure, we relaxed the minimum depth constraint on the first two children of the root, or iterate node, and re-applied the constraint on the third child.

Our individual tree structure also made it necessary to modify the creation process during *full* creation. Full creation is where the resulting tree is constrained to be a "full" tree, or a tree where all branches have the same depth. In figure A.1, the entire tree is not full, but the subtree rooted at "+", is a full tree. With our tree structure constrained to a root node with two terminals and one function, a full tree is not possible. Therefore, we relaxed the constraint for the tree to be full for the first two children (the terminals) of the root node, and re-applied this "full" constraint on the third child, the program to execute at each step in the iteration process.

We also modified the creation process to check to make sure that a newly created individual does not already exist in the population. While the probablity of creating identical individuals is relatively small, we want to ensure that our initial population is as diverse as possible.

Another modification to the Open BEAGLE framework is the addition of the ability to *seed* the population. Seeding allows a hand-written or previously evolved individual to be added to an otherwise randomly created initial population. BEAGLE did not previously support seeding, and with Martin's and our work exploring the use of seeding to boost performance, it became a necessary feature to add. We implemented seeding such that the seed individuals are put into BEAGLE's configuration file within an XML tag, <SEEDS></SEEDS>. With the seeds specified in this way, we modified the base class of all initialization operators to do the following:

1) Check the configuration file for any seeds to insert into the initial population.

2) If there are one or more seeds, read them from the configuration file and insert them into the population.

3) Finish filling the population with randomly generated individuals.

This seeding mechanism was included in version 2.1.0 of Open BEAGLE and was the start of our cooperative development with the BEAGLE creator.

Appendix B

Example individual

See next page.

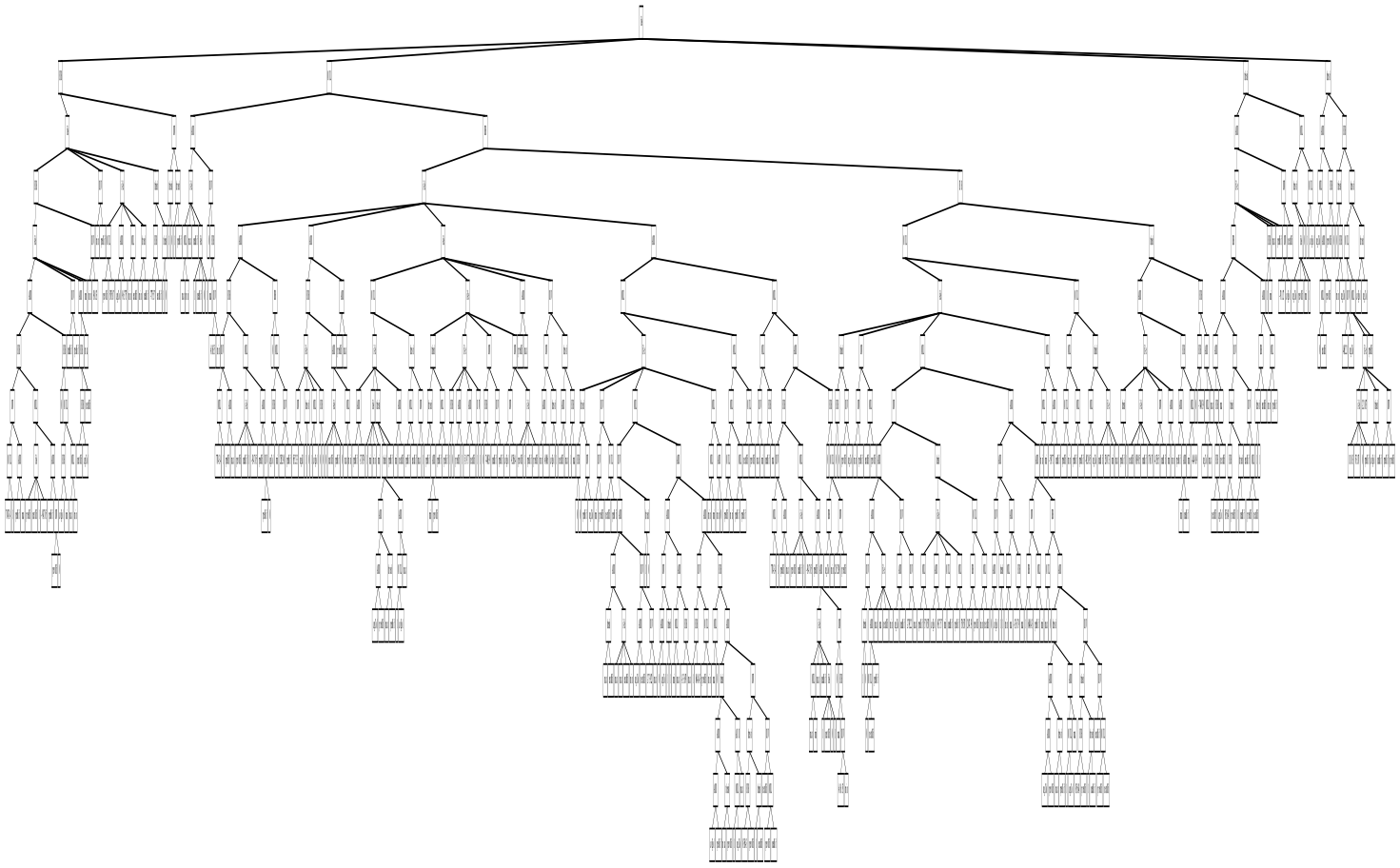


Figure B.1: An example individual with 687 nodes from the laser only runs. While the nodes are unreadable, this figure indicates the size of the evolved programs.

References

[1] Jean-Marc Alliot, Evelyne Lutton, Edmund Ronald, and Dominique Snyers, editors. *Artificial Evolution*, volume 1063 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1995.

[2] David Andre. Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. In Kinnear Jr. [17], chapter 23.

[3] Walter Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, CA, 1998.

[4] Stefano Cagnoni, Riccardo Poli, George Smith, David Corne, Martin Oates, Emma Hart, Pier Luca Lanzi, Egbert Jan Willem, Yun Li, Ben Paechter, and Terence C. Fogarty, editors. *Real-World Applications of Evolutionary Computing*, volume 1803 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 2000.

[5] Dave Cliff, Phil Husbands, Jean-Arcady Meyer, and Stuart W. Wilson, editors. *From Animals to Animats: Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, Cambridge, MA, 1994. MIT Press.

[6] Bruce A. Draper, Jose Bins, and Kyungim Baek. ADORE: Adaptive object recognition. *Videre: Journal of Computer Vision Research*, 1(4), 2000.

[7] Joe Dumoulin, James A. Foster, James F. Frenzel, and Steve McGrew. Special purpose image convolution with evolvable hardware. In Cagnoni et al. [4], pages 1–11.

[8] Terence C. Fogarty, editor. *Evolutionary Computing*, volume 1143 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1996.

[9] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[10] Christopher T. M. Graae, Peter Nordin, and Mats Nordahl. Stereoscopic vision for a humanoid robot using genetic programming. In Cagnoni et al. [4], pages 12–21.

[11] Inman Harvey, Phil Husbands, and Dave Cliff. Seeing the light: Artificial evolution, real vision. In Cliff et al. [5], pages 392–401.

[12] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 2nd edition, 1975.

[13] Ian Horswill. Polly: A vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 1993.

[14] P. V. H. Hough. A new method and means for recognizing complex pattern, 1962. U.S. Patent 30690653.

[15] X. Jiang and H. Binke. Vision planner for an intelligent multisensory vision system. In *Automatic Object Recognition IV*, pages 226–237. 1994.

[16] Michael Patrick Johnson. Evolving visual routines. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[17] Kenneth E. Kinnear Jr., editor. *Advances in Genetic Programming*. MIT Press, Cambridge, MA, 1994.

[18] John Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[19] John Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, 1994.

[20] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis*. Morgan Kaufmann, San Francisco, CA, 1999.

[21] Evelyne Lutton and Patrice Marintez. A genetic algorithm with sharing for the detection of 2d geometric primitives in images. In Alliot et al. [1], pages 287–303.

[22] Pattie Maes, Trevor Darrell, Bruce Blumberg, and Sandy Pentland. The ALIVE system: Full-body interaction with animated autonomous agents. In *Proceedings of the Computer Animation Conference, Geneva, Switzerland*. IEEE Press, 1995.

[23] Martin C. Martin. *The Simulated Evolution of Robot Perception*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2001.

[24] Riccardo Poli. Genetic programming for feature detection and image segmentation. In Fogarty [8], pages 110–125.

[25] Astro Teller and Manuella 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.

[26] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.

Vita

Andrew J. Marek

Date of Birth January 25, 1980

Place of Birth Manchester, Connecticut

Degrees B.S. Summa Cum Laude, Computer Science, May 2002
M.S. Computer Science, May 2004

Professional Societies American Association for Artificial Intelligence

Publications

Marek, A. J., Smart, W. D., and Martin, M. C. (2003). Visual Feature Detection using Genetic Programming, *IEEE Workshop on Learning in Computer Vision and Pattern Recognition.* Madison, Wisconsin, ISBN:0-7695-1900-8.

Marek, A. J., Smart, W. D., and Martin, M. C. (2002). Learning Visual Feature Detection for Obstacle Avoidance Using Genetic Programming, *Genetic and Evolutionary Computation Conference: Late-Breaking Papers.* New York, New York, pp. 330-336.

May 2004