

Genetic Programming for Robot Vision

Martin C. Martin

Artificial Intelligence Laboratory
Massachusetts Institute of Technology
mm@cmu.edu

Abstract

Genetic Programming was used to create the vision subsystem of a reactive obstacle avoidance system for an autonomous mobile robot. The representation of algorithms was specifically chosen to capture the spirit of existing, hand written vision algorithms. Traditional computer vision operators such as Sobel gradient magnitude, median filters and the Moravec interest operator were combined arbitrarily. Images from an office hallway were used as training data. The evolved programs took a black and white camera image as input and estimated the location of the lowest non-ground pixel in a given column. The computed estimates were then given to a hand-written obstacle avoidance algorithm and used to control the robot in real time. Evolved programs successfully navigated in unstructured hallways, performing on par with hand-crafted systems.

1. Introduction

Computer vision in unstructured environments, such as a typical office environment, is notoriously difficult. Different algorithms have their strengths and weaknesses, and no one algorithm is universally better or worse than the alternatives. In this work, Genetic Programming was used with a representation close to existing, hand-written algorithms to create an algorithm that worked empirically for a particular environment.

As in any field of research, one can find threads in the literature by following the evolution of a single idea. The idea that most inspired this work started with Ian Horswill's Ph.D. thesis on Polly the Robot [5]. Polly gave simple tours of the seventh floor of the MIT AI lab, which had a textureless carpeted floor. Obstacles, or at least their boundaries, could therefore be detected as areas of visual texture. The system had problems with other carpet patterns, or even sharp shadows. Liana Lorigo extended this work by assuming the bottom of each image represented floor, and searched for areas with different colors or texture than the floor [12]. However, an object near the robot

could confuse it. Iwan Ulrich and Illah Nourbakhsh extended the work by taking the floor to be part of a previous image that had since been traversed [30].

In this work, Genetic Programming can be seen as automating this process. A traditional supervised learning framework was used. Images were collected from an office hallway, and the lowest non-ground pixel in six columns of each image was determined by hand. Programs were then evolved that, given an image and the location of a column, estimated the lowest non-ground pixel. An obstacle avoidance algorithm was then constructed by hand that used these estimates to guide the robot.

In [15] I presented an early implementation of this framework along with initial results. That work required that a hand written, poorly performing "seed" individual be inserted in the initial population. This paper describes several refinements that, among other things, obviate the need for the seed individual.

2. Previous Work

2.1 Evolutionary Robotics

Evolutionary Robotics is an emerging field that uses simulated evolution to produce control programs for robots. Recent work can be found in [7] and [23]. Most work uses bitstring Genetic Algorithms to evolve neural nets for obstacle avoidance and wall following using sonar, proximity or light sensors, e.g. [8, 20, 19]. Significantly, gradient based learning techniques consider recurrent neural networks much harder to train than feed forward networks, since gradient information typically isn't available. Evolutionary Computation doesn't use gradient information, and therefore even exploratory, toy problems can use recurrence.

Genetic Programming has been used occasionally. Lee et al. [11] use GP to evolve behavior primitives and arbitrators for a behavior based mobile robot. Nordin et al. [24] use a Genetic Programming variant that directly manipulates SPARC machine language. They use symbolic regression to predict the "goodness" of a state 300 ms in the future, based on the current sensor readings and

action. For obstacle avoidance, the goodness is simply the sum of the proximity sensors, plus a term to reward moving quickly in a straight line. To choose a direction, the robot runs the best individual for all possible actions with the current sensor readings. The action with the highest estimated goodness is chosen. They use a population size of 10,000 and find that, in runs where perfect behaviour developed, it developed by generation 50.

Most work evolves in simulation, with the best individuals then run on a robot in the real world. Reynolds [27] has pointed out that without adding noise to a simulation, EC will find brittle solutions that wouldn't work on a real robot. Jakobi et al. [8] discovered that if there is significantly more noise in the simulation than on the real robot, new random strategies become feasible that also don't work in practice.

As well, to my knowledge, no one has tried to simulate CCD camera images, either using standard computer graphics techniques or morphing previously captured images. The Sussex gantry robot [2] uses a CCD camera, but the images are reduced to the average brightness over three circles. These are significantly easier to simulate than a full CCD image, especially when the only objects are pure white on a black background. Smith [28] simulated a 16 pixel one-dimensional camera with auto iris on a robot soccer field. The 16 pixels were actually derived from 64 pixels; Smith doesn't say how. This is an important step, but again much easier than simulating a CCD image at, say, 160 x 120 pixels or above.

A few research groups perform all fitness evaluations on the real robot. Floreano and Mondada [4] evolve recurrent neural networks for obstacle avoidance and navigation from infrared proximity sensors. It takes them 39 minutes per generation of 80 individuals, and after about 50 generations the best individuals are near optimal, move extremely smoothly, and never bump into walls or corners. Naito et al. [22] evolve the configuration of eight logic elements, downloading each to the robot and testing it in the real world. Finally, the Sussex gantry robot [2] mentioned earlier has used evaluation on the real robot. They used a population size of 30, and found good solutions after 10 generations.

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

In summary, Evolutionary Robotics has used low bandwidth sensors, such as sonar or proximity sensors, presumably to cut down the amount of information to process. There are typically less than two dozen such sensors on a robot, and each returns at most a few readings a second. However, much traditional work in computer perception

and robotics uses video or scanning laser range finders, which typically have tens to hundreds of thousands of pixels, and are processed at rates up to 10 Hz or more. Evolutionary Robotics has much to gain by scaling to these data rich inputs.

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

2.2 Visual Obstacle Avoidance

Somewhat surprisingly, there have only been a handful of complete systems that attempt obstacle avoidance using only vision in environments that weren't created for the robot. Larry Matthies' group has built a number of complete systems, all using stereo vision [17]. They first rectify the images, then compute image pyramids, followed by computing "sum of squared differences", filtering out bad matches using the left-right-line-of-sight consistency check, then low pass filter and blob filter the disparity map.

Their algorithm has been tested on both a prototype Mars rover and a HMMWV. The Mars rover accomplished a 100m autonomous run in sandy terrain interspersed with bushes and mounds of dirt [16]. The HMMWV has also accomplished runs of over 2 km without need for intervention, in natural off-road areas at Fort Hood in Texas [18]. The low pass and blob filtering mean the system can only detect large obstacles; a sapling in winter, for example, might go unseen.

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

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

Liana Lorigo's algorithm [12, 13] assumes the bottom of the image represents clear ground, and searches up the

image for the first window that has a different histogram than the bottom. This is done independently for each column. If the ground is mostly flat, then the further up the image an object is, the further away it is. The robot heads to the side (left or right) where the objects are higher up.

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

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

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

Ulrich and Nourbakhsh [30] took the floor to be the part of a previous image that had since been traversed.

Illah Nourbakhsh has used depth from focus for robot obstacle avoidance [25]. Three cameras, focused at different distances (near, middle and far), image the same scene. Whichever image is sharpest is the most in focus, so the objects are roughly at that distance. Actually, the images are divided into 40 windows (8 across and 5 down), which are treated independently, giving an 8 by 5 depth map.

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

3. Framework

Previous evolutionary robotics work has incorporated video as a sensor, but to do so has forced the image through a huge bottleneck, to either three or sixteen pixels. With such a low resolution, only problems in carefully constructed worlds were possible.

For the robot to be truly embodied in a real world environment, and to avoid all manner of obstacles using only vision, requires more complex algorithms that respond to more details in the image. Simulating real images to the required fidelity would be a research project in itself, if even possible at the speeds needed to support simulated evolution. However, evolving on the real robot is also not possible, since this limits the number of evaluations, and hence the complexity of what can be evolved. The research groups that have attempted this use population sizes of less than 100 individuals, whereas GP typically uses sizes of 2000 to 10,000.

Record Video

Robot, Real Time

Learn Offline

Genetic Programming

Build Navigation

By Hand

Validate Online

Robot, Real Time

Figure 1: The four phases of constructing the robot control program. First, a set of representative images were collected. Then an offline genetic algorithm created a vision subsystem. Next, a navigation subsystem was written by a human programmer. Finally, the combination of these two was used to control the robot.

In addition, without specifying any *a priori* structure for the evolved programs, the problem becomes many orders of magnitude harder than previous work. The input, essentially the amount of light in various directions, is simply too distantly related to the output, i.e. the direction to travel.

For both these reasons, the control program was divided into two components, the *vision subsystem* and the *navigation subsystem*. The interface between the two was completely specified: the vision algorithm takes in an image and a direction, and returns an estimate to the nearest obstacle in that direction. The navigation algorithm starts with a number of such estimates from the current image, and computes a direction to travel.

While specified, this interface is not arbitrary. Sonar and laser range finders, by far the most popular sensors, return distances, and the number of such estimates per image, six for training and twelve while controlling the robot, are a minimal representation of the environment compared to the dense depth map typically used in stereo based navigation. This minimal representation was inspired by arguments that representation should be used sparingly. The number six was chosen to be roughly equal to the number of sonar sensors in the camera's roughly 90° field of view. During online validation six columns was found to be too few, but twelve was sufficient. No other numbers were tried.

The vision subsystem was constructed first, independent of the navigation subsystem, see Figure 1. Then the navigation algorithm was constructed to be robust to the sorts of errors made by the vision algorithm. Lessons learned about which errors are easy to compensate for and which are more difficult could guide the design of the next version of the vision system.



Figure 2: What the vision subsystem computed. For a given column of the image, the evolved vision subsystems computed the boundary between the ground (lower line) and a non-ground object (upper line). If there was more than one such boundary, the lowest was desired. This was done independently for six columns in the image.

For the person constructing the system, there is another asymmetry between the two subsystems: the vision subsystem is much harder to create. The output of the vision subsystem is the estimated distance to the first obstacle in various directions, and in this way similar to sonar. The navigation subsystem can therefore adapt designs created for sonar. When it comes to navigation, sonar is a much more common sensor than vision, so navigation from this type of data is well understood. In contrast, while there has been much work on computer vision, there has been little applying it to obstacle avoidance on a mobile robot.

For this reason, the core of this work was the development of the vision subsystem, which was constructed using Genetic Programming [9]. Potential programs were given an image and column location as input, and produced a single real-valued output, which was interpreted as the pixel location of the lowest non-ground pixel within that column. (See Figure 2.) Assuming objects touch the ground and that the ground is roughly flat, the height of the lowest non-ground pixel is a monotonic function of the distance from the robot. No state was maintained from one image to the next or between columns within an image, which means all programs were reactive.

A traditional supervised learning framework was used. To collect a training set, the robot navigated using sonar and passively collected a video stream. In each of six columns of each image, the correct answer, i.e. the location of the lowest non-ground pixel, was determined by the author and recorded.

An evolved program got a column “correct” if it was within 10 pixels of the hand labelled correct answer. Its fitness was simply the number of correct columns on the entire training set. This differed from previous work [15] where the fitness was the absolute value of the difference between what the individual returned and the hand labelled “correct” answer. The change discouraged the

evolution from focusing differences of a few pixels, and instead focused it on getting more columns in the ballpark, which was much more important for obstacle avoidance.

Table 1: Functions and Terminals

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

The most general and straight forward way of incorporating image information into GP is to simply allow it to access individual pixels and give it some looping constructs. However, with the limited computing power of today’s computers, it would take a long time to find even a simple algorithm that examines the correct location in the image.

To give the representation a little more structure, successful visual obstacle avoidance algorithms were examined and a common building block was found. In all these existing system the bulk of run time was spent performing a computation over a rectangular window which iterated over a column or row of the image.

Thus, a new type of node was created, an *iterate* node, which moved a rectangular window over the image. In a simplification of previous work [15], it only moved vertically and took three arguments: the size of the rectangle, the column in which to iterate, and finally a piece of code to execute at each location. This last piece of code had access to the results of various image operators over the window.

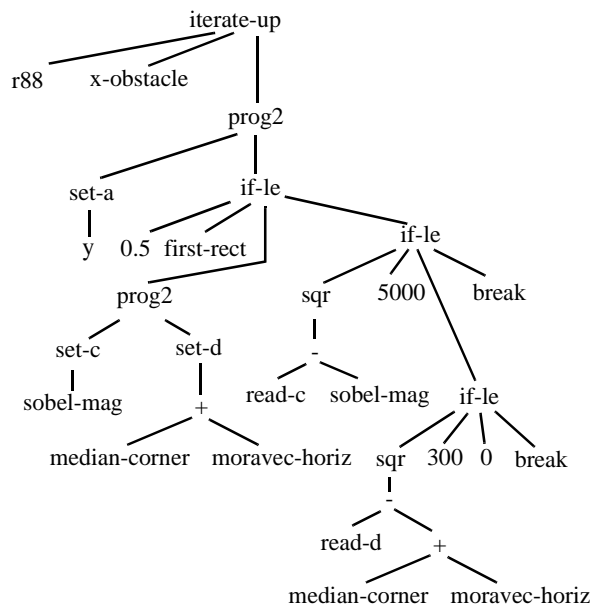


Figure 3: An example window iteration branch, implementing a Lorigo style algorithm [13].

In addition to the *iterate* node and its associated image operator nodes, standard mathematical functions were provided, as well as flow control and five floating point memory registers, with associated read and write nodes, similar to Teller and Veloso’s work with PADO [29]. The complete list of functions and terminals in the window iteration branch is shown in Table 1. An example iterated window branch, implementing a Lorigo style algorithm, is shown in Figure 3.

There was only a single iterate node per tree, always appearing at the root. The creation and crossover operators enforced this constraint. Each individual had two such trees, to allow individuals to make two passes, e.g. to first compute an average intensity of pixels known to be floor, then on the second pass to compare each window to that average. The iterate node did not have a return value. Instead, one trees was designated as the *result producing branch*, and the final value of the first memory location was used at the return value for the entire individual. This tree could also use the final values of the memory locations of the other trees.

This paper extends work that was reported in [15]. There, a hand written, poorly performing “seed” individual was added to the initial population. This seed performed a little better than the best randomly generated individuals in that initial population, and evolution modified many aspects of it, improving its performance greatly.

While many aspects of the seed were modified, others were not. In particular, the successful evolved algorithms all iterated vertically, never horizontally, and from the bottom of the image to the top or vice versa, never starting

part way up. Therefore, horizontal iteration was eliminated and the iterate node 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.

4. Experiments

4.1 Experimental Setup

All experiments were performed on the Uranus mobile robot, in the Mobile Robot Lab at Carnegie Mellon University. The robot has a three degree of freedom base with dead reckoned positioning. While forward/backward motion and turning in place are fairly accurate ($\sim 1\%$ error), sideways motion isn’t (about 10-20% error, significant rotation). For sensing it uses a b/w analog video camera and a ring of 24 sonar sensors. Processing was done by an off board 700 MHz Pentium III computer running BeOS.

Data was collected from two hallways in different buildings. The most problematic features were glossy, textureless grey walls which often confound local depth estimation techniques such as stereo, optical flow and depth from focus.

To collect images that were representative of what the cameras would see, the robot collected data while avoiding obstacles using sonar. While obstacle avoidance using sonar is considered easier than using vision, it still took many attempts to get a working system. The method that proved most successful determined speed from proximity to the nearest object, and determined direction of travel by fitting lines to points on the left and right sides of the robot. More details can be obtained from [14]. The data is summarized below.

The camera was calibrated using the system described in [21]. The camera was then mounted on the robot, pitched 51 degrees down from horizontal, and images of size 320 by 240 pixels were collected. These images became the training sets. Ground truth was then assigned by hand using a simple GUI. Typical images and ground truth are shown in Figure 4.

NSH Hallway

Total Number of Frames:	328
Frames In Training Set:	65 (every fifth)
Number of Fitness Cases:	$65 \times 6 = 390$
Elapsed Time:	75 sec
Frame Rate Of Training Set:	$65 \div 75 = 0.87$ fps

While the robot had to travel mostly straight down a hallway, it started out a little askew, so it approached one side. At one point, a person walks past the robot and is clearly visible for many frames. At the end of the hallway

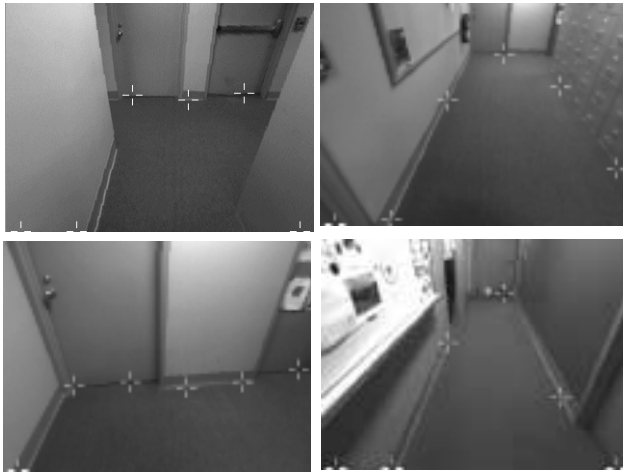


Figure 4: Training data, with ground truth indicated.

it turns right. The carpet is grey with a large black stripe at one point. The shadow of the robot is visible at the bottom of most frames.

FRC Hallway

Total Number of Frames:	356
Frames In Training Set:	71 (every fifth)
Number of Fitness Cases:	$71 \times 6 = 426$
Elapsed Time:	82 sec
Frame Rate Of Training Set:	$71 \div 82 = 0.87$ fps

Starts with robot in lab doorway. Moves straight until its in the hallway, then turns right, travels down hallway, at end turns right, then travels straight to dead end. All doors were closed. The fluorescent light bulbs at the start are burned out, so the intensity of the carpet varies widely. The shadow of the robot is visible at the bottom of most frames.

Combined

Total Number of Frames:	$328 + 356 = 684$
Frames In Training Set:	68 (every tenth)
Number of Fitness Cases:	$68 \times 6 = 408$
Elapsed Time:	$75 + 82 = 157$ sec
Frame Rate Of Training Set:	$68 \div 157 = 0.43$ fps

This data set was simply the combination of the above two data sets, using every tenth frame instead of every fifth in order to keep the training set size approximately equal.

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

Genetic programming was performed on a dual 700 MHz Pentium III, and evaluation times varied widely, but averaged approximately 725 msec. The time for a simu-

Table 2: Koza Style Tableau

Objective	Given an image and a horizontal position within it, return the first non-ground pixel in that column.
Architecture of individuals	Two <i>window iteration</i> branches.
Function and Terminal Sets	See Table 1
Fitness cases	Six columns in each of 65-71 images. 390-426 total.
Raw fitness	The percentage of fitness cases “correct.” A fitness case was “correct” if the pixel location estimated by the evolved program was within 10 pixels of the author’s determination of the “correct” answer.
Standardized fitness	100% minus raw fitness.
Parameters	51 generations, population size of 10,000, tournament selection with tournament size of 7, ramped-half-and-half with min size 6 and max size 9.

lated evolution run also varied widely, averaging approximately 20 hours. Ten runs were completed in each of the three experiments. A Koza style tableau is shown in Table 2.

Each evolved program took as input the image and the horizontal position of a column. It returned a single number, the vertical position, in pixels, of the first (i.e. lowest) non-ground pixel in that column of that image. It was run on six different columns per image. As mentioned earlier, a program’s performance on a given column of a given image was deemed “correct” if its answer was within 10 pixels of the human-provided answer, and the total fitness was simply the number of correct columns in the training set. In contrast to earlier work, no seed individual was needed.

After the offline learning, a hand written navigation algorithm used the estimates to decide speed and direction to travel. This algorithm was very similar to the one used during data collection, except that its inputs came from vision, not sonar. The three different vision subsystems, one from each experiment, all used the same navigation subsystem. The best evolved algorithm was run on twelve columns in the image, twice as many as used in training.

The navigation algorithm classified estimates as either near (requiring an immediate halt), medium (slow to 2/3 speed to avoid collision) or far (avoid them before they

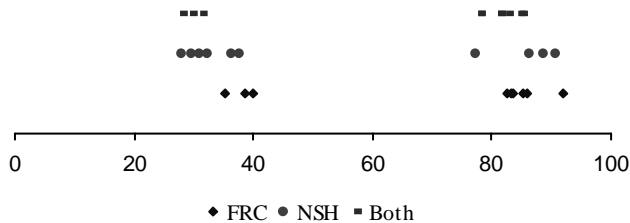


Figure 5: A scatter plot showing the fitness, i.e. the percentage of fitness cases correctly estimated, of the best-of-un individuals in all three experiments.

become a problem.) This case based approach is inspired by the Property Mapping approach of Nourbakhsh [26]. If any of the middle four estimates are in the lower fifth of the image, or either of the two readings outside are at the bottom, then the object is considered near and the robot immediately halts. Otherwise, it looks for objects within four feet to the left and the right of where the robot is and where it would be if it continued straight. To convert pixel height in the image to real world distances, it assumes that the floor is flat, and that the non-ground object touches the floor. If an object is sighted, on either the left or the right, a line is drawn through the readings on each side, and the robot turns to run parallel to the lines.

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

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

4.2 Training Results

This section is necessarily brief. For more details, the reader should consult [14]. A scatter plot of the final individuals in all conditions is show in Figure 5. All three experiments produced individuals which achieved a fitness of greater than 85%. They did this despite burned out lights and other effects that caused the carpet’s average intensity to vary from zero to at least 140s out of 255; despite large gradients caused by imaging artifacts; despite moiré patterns of image noise; and despite the shadow of the robot.

In the Field Robotics Center, the best constant approximation, ignoring both the image and the desired column,



Figure 6: Performance on the training data of the best individual from the FRC hallway experiment.

was to return $\text{image-max-y} - 10$, i.e. 228, to get 101 answers correct for a fitness of 23.7%. In all but two of the runs, the best individual in the initial population did worse than that, returning $\text{image-max-y} - 3$ and achieving 21.6%. The other two runs achieved 29.3% and 36.4% in the initial population. The other two conditions were similar, occasionally but rarely performing better than the best constant approximation.

As can be seen in Figure 5, the results of all three experiments were bimodal. Interestingly, if a run was going to end up in the higher scoring mode, the fitness of the best-of-generation individual was already above 50% by generation 11.

The best individual from all FRC runs achieved 91.78%. Some examples from the training set are shown in Figure 6. While hard to see, the rectangles in the figure are the same size as used by the individual. In the two upper images, all six fitness cases were determined correctly. In the lower left, one of the two on the filing cabinets was higher than desired, although a difference this small would not affect obstacle avoidance significantly. In the lower right, the rightmost column should be at the bottom of the image instead of near the top.

The other two conditions did almost as well, with the best individual from all NSH runs achieving 90.51%, and the best from the combined data set 84.8%. In all three experiments the errors were transient with the exception of the stripe of red carpet in Newell Simon Hall, which was uniformly considered an obstacle, at least when near the camera.

The best individual from the FRC experiment was simplified using a combination of programming tools and by hand. The resulting program is quite readable, see

Result Producing Branch:

Iterate-Down, 3x3 window, centered on desired column:

```
if y ≤ 9 then
  a := y;

if second-branch-b > 9 then
  a := y;

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

  if gradient > 239 and (gradient / 239)4 > diag-grad then
    a := y;
}
```

Second Branch:

b (initial value) := 3.40282e+38

Iterate-Up, 3x3 window, centered on desired column:

$$b = \frac{b(1+h) - (1+h+h^2+h^4)\text{desired-x}}{h^5}$$

Where h is horizontal gradient.

Figure 7: A simplified version of the best individual from the FRC experiment.

Figure 7. The two branches represent two very different styles of algorithm. The result producing branch was essentially a decision tree, although one of the decisions is a non-linear boundary in the space spanned by two image operators. The other branch, which detects obstacles at the bottom of the image, is a recurrent mathematical function involving both a single image operator and the location of the column.

This division into two cases—gradient based boundaries and objects that touch the floor—was discovered automatically. Nothing in the representation suggested the two different cases, nor that the two branches should each tackle a separate case. Similarly, ignoring the gradient when the image intensity was low, which ignored artificial gradients caused by a quirk in the imaging process, was not suggested by the representation either. These are examples of the genetic algorithm simultaneously exploit-

ing regularities in both the problem domain and the representation.

The other best-of-experiment individuals were similarly simplified to discover how they worked. Evolution had exploited a number of techniques, including a sequence of if-then conditions similar to a decision tree but involving non-linear combinations of up to five different image terminals. In all cases, the bottom of the image was handled using different code than the rest of the image. This reflects a natural dichotomy in the images: at the bottom of the image a program must detect the presence or absence of an object, but in the middle of an image it must detect the *transition* between ground and non-ground. During a transition there is likely to be a significant gradient, whereas at the bottom of the image there isn't. The mechanisms for this varied; in the FRC experiment it used two different branches for the two conditions, whereas in the NSH and combined experiments a multitude of *if* statements were used to run different code at different locations. Interestingly, the raw image was never used, although the median filtered image was. All directional gradients of the image intensity were used except the vertical.

4.3 Validation on the robot

All three individuals generalized surprisingly well when run on new video from the same camera in the same hallway. They were relatively insensitive to the pitch and the horizontal location of the column in the image. With the camera set to automatic gain they also provided acceptable results over a wide range of iris settings. They detected objects that weren't present during training, such as chairs or people, with only a little less fidelity than they detected walls. They were also fast, running at about 10 Hz on a 700 MHz Pentium III.

The three algorithms were more than good enough for navigation. They produced occasional glitches, most often declaring that a pixel at the bottom of the image was non-ground when it was, in fact, ground. To stop these from causing too many panic halts, the hand written navigation algorithm filtered readings by taking the minimum (highest pixel location) of consecutive estimates.

The robot was then run in the same environment(s) it was trained in. Videos of this *online validation* can be found at www.metahuman.org/martin/Dissertation. These routes retraced the path of the training set and then continued much further. They included views of the same hallway from the opposite direction, as well as similar areas never seen during training. Objects were present that were not present during training, such as chairs, trash cans and people.

In general, the navigation system worked rather well. The evolved algorithms worked well despite months of

wear & tear on the carpet. Most errors were transient, lasting 1 or 2 frames, even when the camera was stationary. It worked on a range of iris settings and camera tilts. It did not overfit to column location, and twice as many columns were used during navigation as were used for data collection.

When navigating under sonar, fourteen sonar sensors were used for a total field of view of approximately 215 degrees, seeing well to the sides. The area of awareness with vision was much smaller than with sonar, and entirely in front of the robot's base. In the reactive framework described here this makes it almost impossible to successfully navigate doorways, especially since the robot is only a few inches narrower than them. However, it performed very well at corridor following and avoiding obstacles such as people and chairs.

There were a few persistent errors. It was sensitive to small strips of paper or shiny pieces of metal. The red carpet, which was handled poorly even on the training data, was classified as an obstacle when nearby, causing the navigation to consistently treat it as a wall. Other errors would fool the navigation system only occasionally, although these were responsible for most collisions.

In 5 validation runs, the mean distance and time to collision for the FRC individual was 133 meters, 20:01 minutes, and the longest was 260 meters, 39:32 minutes. These are comparable to the published performance of the hand-written systems described in the Previous Work section.

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

5. Bibliography

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

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

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

[4] D. Floreano and F. Mondada. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-

Network Driven Robot. *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*. 1994.

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

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

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

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

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

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

[11] W.P. Lee, J. Hallam and H.H. Lund Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots. In *Proceedings of IEEE 4th International Conference on Evolutionary Computation*, IEEE Press, 1997.

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

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

[14] M.C. Martin, *The Simulated Evolution of Robot Perception*, Ph.D. Dissertation and Carnegie Mellon University Technical Report CMU-RI-TR-01-32. 2001, 159 pp.

[15] M.C. Martin Visual Obstacle Avoidance using Genetic Programming: First Results, *Proceedings of the Genetic and Evolutionary Computation Conference*, July 7-11, 2001, pp. 1107-1113

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

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

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

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

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

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

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

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

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

[25] I. Nourbakhsh, A Sighted Robot: Can we ever build a robot that really doesn't hit (or fall into) obstacles? *The Robotics Practitioner*, Spring 1996, pp. 11-14.

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

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

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

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

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