Final pre-publication draft of:

An Evolved, Vision-Based Model of Obstacle Avoidance Behavior

by Craig W. Reynolds

author's *current* address:
cwr@red3d.com (as of November 27, 2002)
http://www.red3d.com/cwr/
~~Electronic Arts~~
~~1450 Fashion Island Boulevard~~
~~San Mateo, CA 94044~~
~~USA~~

~~phone: 415-513-7442~~
~~fax: 415-571-1893~~

~~email: creynolds@ea.com~~
~~(or cwr@red.com)~~

Submitted for publication in:

Artificial Life III Proceedings
Santa Fe Institute Studies in the Sciences of Complexity
Proceedings Volume XVI

Search for: [???]

Note: code is in font: `Monaco 12`

Craig W. Reynolds
Electronic Arts

An Evolved, Vision-Based Model of Obstacle Avoidance Behavior

Abstract

Using a simple computational model of visual perception and locomotion, obstacle avoidance behavior can emerge from evolution under selection pressure from an appropriate fitness measure. The Genetic Programming paradigm is used to model evolution. Both the structure of the visual sensor array, and the mapping from sensor data to motor action is determined by an evolved control program. The motor model assumes an innate constant forward velocity and limited steering. The agent can avoid collisions only by effective steering. Fitness is based on the number of simulation steps the agent can run before colliding with an obstacle.

Introduction

In the work described here, a behavioral controller for a 2d *critter* (artificial animal, autonomous agent, robot, or what-have-you) is obtained through simulated evolution. A stimulus-response controller for a *vision-based obstacle avoidance* task can spontaneously emerge under selection pressure from a fitness measure relevant to the task. The evolutionary process starts with primitive computational elements that provide simulated perception and motor control, as well as the arithmetic and logical connections between them. The fitness of an individual controller is determined by installing it in a simulated critter in a simulated world and judging the performance it achieves there. The critter moves forward at a constant rate and the controller must "steer" to avoid collisions with obstacles in its simulated environment.

Collisions are considered fatal. A critter's fitness is based on how long it can run before hitting an obstacle. Over time, under the patient guidance of fitness testing, the evolutionary process constructs an increasingly effective mapping from perception to motor control which allows the critters to get increasingly effective at avoiding collisions.

This work is not intended to be a realistic model of the evolution of vision-based obstacle avoidance in natural animals. Instead it provides an abstract example of how vision-based obstacle avoidance *can* arise in an evolutionary process. It provides a computational model for examining theories about how specific selection pressures and various environmental factors affect the evolution of obstacle avoidance strategies.

While admittedly unconnected to natural evolution of corresponding behaviors in real animals, this work aspires to a certain level of plausibility by the *closed* nature of its simulated world. The controller's action is fully determined by the information it obtains about the simulated world through its simulated perceptions. The fitness of a controller is fully determined by the performance of its simulated behavior in the simulated world. The critter's behavior is *grounded* in its perception of the world, and its perception directly reflect the consequences of its behavior.

Maja Mataric has observed that "obstacle avoidance is what robots spend most of their time doing" [Mataric 1992]. It is only after these seemingly simple behaviors are correctly handled that the presumably more complex behaviors can be built on top of them. That obstacle avoidance has such an important and central role may seem counterintuitive. We humans pay scant attention to our own obstacle avoidance because it happens on such a low, almost subconscious, level. To become (painfully!) aware of how much time we spend engaged in active vision-based obstacle avoidance, we

need only close our eyes while walking through an unfamiliar place to see how many obstacles we "discover."

One approach to evolving a sensor-based obstacle avoidance strategy is to postulate a certain fixed sensor array, specifying the number of sensors and their geometric configuration. Then the problem becomes the evolution of a control structure which maps those sensor channels into motor action. This approach would be appropriate if for example we were dealing with a pre-existing robot vehicle and wanted a controller for it. In contrast, natural animals *coevolve* their visual systems and motor control systems over the millennia, neither is fixed in advance. One goal of this project was to include this kind of coevolution in the model. Part of the motivation was a bias for a "natural" as opposed to "robotic" model. Another motivation was a concern that dictating sensor density or placement would indirectly dictate which obstacle avoidance strategy would emerge. The hope was that the unifying concept of simulated Darwinian evolution could simultaneously solve both the sensor placement problem and the obstacle avoidance problem. The results presented here show will demonstrate that such coevolution can occur.

Previous Work

Because the work described here overlaps with many areas of active research, the amount of related previous work is enormous. This section is an attempt to identify some of the most closely related work, with special emphasis on work that lead to or inspired the current project. While all previous work on obstacle avoidance in autonomous agents is relevant to some degree, the work described here is concerned with behaviors that are: *evolved* (as opposed to hand-coded), *remote sensor based* (rather than using touch sensors, or global knowledge of the world), and based on *variable*

sensor morphology (instead of static placement of a predefined number of sensors).

A classical work in behavioral modeling is [Braitenberg 1984] which touched on many of the ideas here, but as thought experiments whose implementation was somewhat fanciful. Many researchers (for example [Brooks 1986] and [Nehmzow 1989]) have worked with simple obstacle avoidance for real robots based on touch sensors.

Obstacle avoidance techniques have been developed for synthetic actors in computer animation: [Reynolds 1987], [Ridsdale 1987], [Reynolds 1988], [Amkraut 1989], [Ridsdale 1990], and [Girard 1990]. All of the se were based on global knowledge about the environment, which while suitable for their intended application, is an unnatural model for an autonomous agent.

In contrast, [Renault 1990] developed a hand-coded, vision-based obstacle avoidance technique for animation applications. In this system the agent examined its simulated environment by a variant of computer graphic *rendering*, producing a z-buffer (range image) in addition to a color perspective image. This allowed the agent to direct its motion based on information about the distance to various obstacles in its path. Renault's system made use of "object tags" which allowed it to unambiguously determine what object was visible at each pixel and how far away it was. This too can be seen as a form of global knowledge, or as imposing a constraint on the environment, such as requiring every object to be painted a unique color.

An example of non-evolved, fixed-sensor architecture, obstacle avoidance for real time robotics based on video image processing is presented in [Horswill 1991]

Evolved vision-based behaviors are central to PolyWorld [Yaeger 1993], a conceptually vast, open-ended Artificial Life simulator. In PolyWorld, as in nature, creatures thrive to the extent

that they are able to find food and mate. Unlike a Genetic Algorithm system, there are no externally-imposed fitness tests in PolyWorld, and no explicit goal other than survival. Simple behaviors like obstacle avoidance can appear in PolyWorld, but only as a side effect of the creature's quest for survival. The simulated world is rendered to provide color pixel data for a one dimensional retina which is the primary input to the creatures behavioral controller.

An interesting example of another kind of evolved behavior is seen in [Ventrella 1990], a legged critter whose interactive, physically-based walking behavior was derived through evolution. The walker's fitness was measured by their ability to move toward a globally designated "food" location. In a similar vain [Maes 1990] describes robotic leaning of coordinated leg motions, and [de Garis 1990] used a mutation-based technique to produce controllers for leg motion. In both cases, fitness was based on achieving forward motion.

The behavioral controllers described in this paper are very much in the spirit of the *subsumption architecture* originally described by Rodney Brooks [Brooks 1986]. These *reactive agents* base their behavior directly on the world as perceived through their sensors. They have little or no higher cognition and do not bother with complex mental models of their environment, preferring to use "the world is its own map" [Brooks 1991].

The current work is very closely related to recent research by Randy Beer and colleagues. His earlier work [Beer 1990] involved painstakingly duplicating by hand the behavioral repertoire of an insect. His more recent work [Beer 1992] derives weights for continuous-time recurrent neural net behavioral controllers using simulated evolution based on fitness criteria that measure achievement of the desired behavior. A detailed analysis of the complex dynamics these controllers can be found in [Gallagher 1993].

The work reported here was also strongly influenced by Dave Cliff's manifesto on *computational neuroethology* [Cliff 1991a] as well as his hoverfly simulation [Cliff 1991b]. While the models described here are not based on neurons, the principles of using a closed, grounded simulation to test behavioral models are fundamental to this project. Contemporaneous with the work reported here, the Evolutionary Robotics group (Harvey, Cliff, and Husbands) at the University of Sussex were engaged in some very closely related experiments, subsequently reported at SAB92. In [Harvey 1993] they describe evolving dynamical, recurrent neural nets to control a simulated robot using touch sensors engaged in tasks such as "wandering" and "exploring" a space cluttered with obstacles. In [Cliff 1993], dynamical, recurrent neural nets are evolved to control a simulated vision-guided robot whose task was to move to, and remain at, the center of a circular room. These vision-guided robots had exactly two eyes, but various parameters such as placement and "zoom" were under the control of evolution. A significant difference between these projects and the work reported here is that the Sussex group used noise at each node in their neural nets, both to simulate real world conditions and to require evolution to discover robust controllers that do not (can not) depend on coincidental or chance correlations in the simulated world.

At about the same time this author did some experiments, using a model essentially identical to the one described here, to investigate the evolution of coordinated group locomotion behavior [Reynolds 1993]. In those experiments, in order to survive, groups of critters needed to avoid collisions with static obstacles, to coordinate with (not run into) each other, and to flee from a pursuing predator. The results obtained in that can at best be considered preliminary.

Genetic Algorithms, Genetic Programming, and the Steady State

At the heart of the work described here is the notion of simulated evolution. The basic evolution model used here is the venerable Genetic Algorithms ("GA"), originally developed by Holland [Holland 1975]. The Genetic

Algorithm has been widely studied by many authors and applied to a myriad of practical problems in many different fields [Holland 1992]. Over the years many variations on the basic Genetic Algorithms have been proposed. A hybrid of two such variations are used to implement the simulated evolution described in this paper.

John Koza forged a link between the Genetic Algorithm and computer programming technology with his Genetic Programming paradigm [Koza 1989], [Koza 1992]. Genetic Programming ("GP") is a technique for automatically creating computer programs (often, but not necessarily, in the Lisp language) that satisfy a specified fitness criteria. There is a very strong analogy between the operation of the Genetic Algorithm and Genetic Programming, then main difference is in the representation of genetic information: bit strings in GA, fragments of Lisp code in GP. The use of fitness-proportionate (or rank-based or tournament) selection, reproduction, crossover, and mutation are all directly analogous.

One significant difference is that while classic Genetic Algorithms work on fixed length bit strings, Genetic Programming deals with objects of inherently varying size. The complexity of programs created by GP tend to correspond to the complexity of the problem being solved. If simple programs do not satisfy the fitness function, the Genetic Programming paradigm creates larger programs that do. As a result, Genetic Programming does not require that the user know, or even estimate, the complexity of the problem at hand. This was an important consideration for the goal of coevolving sensor arrays and obstacle avoidance strategies. We did not want to specify, for example, how many sensors should be used. Genetic Programming did not require such a specification, instead that implicit parameter could be left to evolve its own preferred value.

(Karl Sims independently developed the idea of using Lisp code as genetic material. He used the concept in combination with a system much like *The Blind Watchmaker* ("BW") [Dawkins 1986] where fitness is based on a human's aesthetic judgment. An analogy which might help clarify the relationships is that GA is to GP as BW is to Sims work. A discussion of crossover and mutation operations can be found in [Sims 1991]. These mutation operations were incorporated into the GP system described in this paper, but following Koza's lead, the experiments described here did not use mutation.)

Gilbert Syswerda described a variation on traditional GA he called *Steady State Genetic Algorithms* ("SSGA") in Appendix A of [Syswerda 1989], an analysis of their performance can be found in [Syswerda 1991] and [Davis 1991]. A similar technique had previously used in classifier systems and is described on pages 147-148 of [Holland 1975]. Darrell Whitley independently discovered this variation and described it in [Whitley 1989]. While the term "steady state" has apparently become accepted, the comparison of "traditional GA" versus "steady state GA" suggests terms like "batch" versus "continuous" to this author.

The basic idea of SSGAs is to do away with the synchronized *generations* of traditional GAs. Instead there is a continuously updated population ("gene pool") of evolving individuals. Each step of the SSGA consists of fitness proportionate (or rank-based or tournament) selection of two parents from the population, creation of new individual(s) through crossover and (occasional) mutation, removal of individual(s) from the population to make room, and finally insertion of the new individual(s) back into the population. An additional requirement is that all individuals in the population are required to be unique. (The step of creating new individuals loops until unique offspring are found.) In the work

reported here, the concept of SSGA was applied to GP to produce a system for "Steady State Genetic Programming."

SSGAs are clearly more *greedy* because they focus more intently on high fitness members of the population. For "easy" problems at least, this implies that a SSGA will find a solution more quickly than a traditional GA. However what "greediness" actually implies is faster convergence of the population. This is good news if the population happens to converge on a global maxima, but it is clearly bad news if the population converges prematurely on a local maxima. The likelihood of being stranded on a local maxima is probably what characterizes a "hard" problem.

Obstacle Avoidance as Genetic Programming

In order to solve a problem with Genetic Programming we must restate the problem in a canonical form. We must specify a list of *functions* and a list of *terminals*. The Genetic Programming paradigm will then evolve programs as hierarchical expressions with the functions applied to subexpressions which are either one of the terminals or (recursively) another such expression. These hierarchies are known variously as "s-expressions", "lists", "Lisp fragments", "parse trees" and so on.

The terminals used in the evolved obstacle avoidance problem are just an assortment of numerical constants: 0, 0.01, 0.1, 0.5, and 2. The list of functions is:

```
+
-
*
%
abs
iflte
turn
look-for-obstacle
```

The functions +, -, and * are the standard Common Lisp [Steel 1990] arithmetic functions for addition, subtraction, and multiplication. Each of them take an arbitrary number of arguments. The function abs is the standard Common Lisp absolute value function which takes one argument. The functions % and `iflte` are suggested in [Koza 1992]. Koza calls % "protected divide", a function of two arguments (A and B) which returns 1 if B=0 and A/B otherwise. The conditional `iflte` combines the standard Common Lisp functions `if` and <= into "if less than or equal". In the implementation described here, `iflte` is a Lisp *macro* which makes this source-level transformation:

(iflte a b c d) → (if (<= a b) c d)

The functions `turn` and look-for-obstacle are specific to the obstacle avoidance problem. Each of them take a single argument, an angle relative to the current heading. Angles are specified in units of *revolutions*, a normalized angle measure: 1 revolution equals 360 degrees or 2π radians. These functions will be explained more fully below, but basically: `turn` steers the critter by altering its heading by the specified angle (which is returned as the function's value). look-for-obstacle "looks" in the given direction and returns a measure of how strongly (if at all) an obstacle is seen.

We must also provide the *fitness function* that the Genetic Programming paradigm will use to judge the fitness of the programs it creates. The fitness function takes one argument, an evolved program, and returns a numerical fitness value. In the implementation described here, fitness values are normalized to lie in the range between zero and one inclusively. A fitness of zero means "totally unfit", a category that can include programs that get errors during execution. A fitness of one signifies a perfect solution to the problem at hand.

Finally there are a few other parameters required to specify a Genetic Programming run. In these experiments the maximum size of programs in the initial random generation is set to 50. The size of the "steady-state gene pool" (which is roughly comparable to the population in a traditional "batch" generation GA) is 1000 individual programs. The mutation rate is zero.

The Critter

The critter model used in these experiments is a computer simulation based on widely-used principles of computational geometry and computer graphics. Its simplicity and abstraction make it an equally good (or by the same token, equally bad) model of a living creature, or a mechanical robot, or simply an abstract synthetic creature.

The critter moves on a two dimensional surface. It is essentially equivalent to the LOGO *turtle* [Abelson 1981]. Its state consists of a position and an orientation. In the accompanying illustrations, the critter's body is depicted as a triangle to indicate its heading. For purposes of collision detection, however, its body is modeled as a disk of unit diameter. Besides the critter itself, the world includes various *obstacles*. In these experiments, the obstacles are represented as line segments. A *collision* is when the critter gets too close to an obstacle. Specifically, whenever the distance from its center point to the closest point on any obstacle is less than one half its body length, a collision has occurred and obstacle avoidance has failed. Collisions are considered to be fatal: a critter's life span is determined by how long it can avoid collisions with all obstacles in the world. Figure 1 shows the critter and the obstacle course named "Bilateral" which was used in all of the experiments described here.

There are two kinds of motor actions in the critter's repertoire: *move forward* and *turn*. In the particular problem being studied here, we will assume that forward motion is constant and innate. The critter will always move forward (that is, along its own heading) by one half of its body length each simulation step. This implies that we are actually dealing with the "*constant-speed* vision-based obstacle avoidance" problem.

Beyond these fixed innate properties, the critter's *volition* comes from its evolved control program. Typically the controller will use the perception primitive `look-for-obstacle` to get information about the environment, do some conditional and arithmetic processing, and then a `turn` based on the result. Most of the effective evolved control programs contain many calls to both `look-for-obstacle` and `turn`.

Two kinds of limitation were placed on the critter's ability to turn. The first is that if the controller specifies an excessive turning angle it will be truncated to a "reasonable" range. In the experiments described here, "reasonable" is defined to be ± 0.05 revolutions (18 degrees or 0.31 radians). Secondly the critter will die ("from whiplash") if it turns too fast. This can happen despite the first limitation because the controller is capable of performing more than one turn operation per time step. The fatal turn threshold used in these experiments is ± 0.075 revolutions (27 degrees or 0.47 radians). The result of these limitations is to give the critter a non-zero turning radius. The motivation is to prevent them from spinning in place. Strictly speaking, spinning in place *is* a valid solution to the obstacle avoidance problem. Because spinning is a degenerate and not particularly interesting solution, it has been ruled out by these limitations on turning angle.

Perception

The `look-for-obstacle` function simulates the critter's perception and so is the controller's only source of information about its world. All adaptive, time-varying behavior must be derived somehow from the variations of these perceptions as the critter moves through the world. When `look-for-obstacle` is called, a *ray-tracing* operation is performed. That is, a geometric ray ("half line") is constructed from the critter's position in the direction specified by the sum of the critter's heading and the argument to `look-for-obstacle`. The intersection (if any) of that ray with each obstacle is calculated. The obstacle whose ray intersection is closest to the critter's center is deemed the *visible obstacle.*

In order to provide the controller with an indirect clue about distances, we have postulated that the critter's world is a very foggy place. Visual stimuli are attenuated by distance. Certainly this aspect of the simulation can be criticized as being *ad hoc* . But the alternatives are daunting: without attenuation, a critter surrounded by obstacles is in a featureless environment, whatever direction it looks it sees an obstacle. Postulating more complicated sensory mechanisms (such as stereo vision or texture recognition) seemed too complex and would have introduced a new crop of *ad hoc* details to be explained away.

The "foggy world" model is somewhat plausible: real fish in murky water face a very similar perceptual realm. The phenomenon known as "aerial perspective" refers to attenuation in air caused by dust and water vapor. Simulated aerial perspective is widely used (in classical painting, photo-realistic computer graphics, and cinematography of miniatures for special effects) to provide a sense of distance and scale. If desired, say for robotic experimentation, technology exists (in the special effects industry) for filling rooms with "smoke" of precisely controlled density. The last rationalization for the foggy world model is that it is only

slightly different in effect from perception based on active sonar as used by bats and dolphins. A similar effect would be obtained on a real robot using narrow-angle photoreceptors (what photographers call "spot meters") measuring diffuse reflections from uniformly colored walls if the only light source was located on the robot itself. The $1/r^2$ falloff of illumination with distance would produce a result very similar to the "foggy world" model presented here.

The value returned from `look-for-obstacle` is a number between zero and one. A value of one would indicate that the obstacle is coincident with the critter, but this does not occur in practice since this would imply a collision had occurred. As the distance between the critter and the obstacle increases, the visual signal drops off quadratically in strength. At a certain threshold value (15 body lengths in these experiments) the signal will have reached zero. Hence a value of zero returned from `look-for-obstacle` indicates that the closest obstacle in the given direction (if any) is more than 15 units away.

In the original conception of this project, it was imagined that evolution would design a sort of *retina* consisting of a series to calls such as `(look-for-obstacle 0.1)` with various constant values used for the argument. Hence evolution would select the "field of view" and density distribution of the retina. As always, evolution turned out to have its own ideas. There was nothing in the problem statement that said the argument to `look-for-obstacle` had to be a constant, and so the controllers soon evolved bizarre mechanisms based on calculating a dynamic value and using that to specify the direction in which to look. For example one species of obstacle avoider seemed to be based on the puzzling expression (look-for-obs`tacle (look-for-obstacle 0.01)`). In hindsight it becomes clear that the correct model is not that of a retina but rather a simple form of *animate vision* [Cliff 1991b] where the controller "aims" its visual sensor at the area of interest.

Fitness

In order to test the fitness of a newly evolved control program, we place the critter in the world, start it running, and measure how long it can avoid collisions with the obstacles. Specifically, the critter is allowed to take up to 300 steps. The number of steps taken before collision, divided by 300, produces the normalized raw score.

The raw score is modified by multiplication by some "style points" (less objective criteria). In order to encourage critters to use generally smooth paths they are penalized for "excessive turning". In order to encourage symmetrical behavior and discourage looping paths, the critters are penalized for "unbalanced turning". Statistics on turning are accumulated during the run, a sum of all turn angle magnitudes, a sum of signed turn angles, and a count of left-versus-right steps. A critter that made the maximum legal turn each time step would receive the "most harsh" excessive turn penalty of 0.5, whereas a critter that never turned would get value of 1.0 (i.e.: no penalty). The penalty for "unbalanced turning" is analogous.

Using a single fitness test will produce a controller for solving the one exact situation represented by the fitness test. However this solution may turn out to be very fragile and opportunistic. It may well have incorporated expectations of incidental properties of the specific fitness test. In the case of obstacle avoidance, means that a controller would evolve that could nimbly negotiate a given obstacle course flawlessly, but be utterly stumped by a slightly different obstacle course, or by starting its run in a slightly different initial placement.

To strive for *robust* obstacle avoidance we need an alternative to a single fitness test. One solution would be to

randomize some aspect of the world, such as obstacle placement or the critter's starting position. This is an appealing approach since fitness trials in nature are effectively randomized. When randomization was attempted in these experiments, it became clear that the noise injected into the fitness values made it very hard for both the experimenter and for simulated evolution to determine if progress was actually being made. Another approach is to have multiple obstacle course for each critter to run, or the critter can make multiple runs through the same obstacle course from slightly different starting positions. These both have the effect of discouraging fragile solutions, but do not introduce uncorrelated noise into the fitness measure. Some other interesting approaches are discussed in the Future Work section below.

Results

The end result of two runs and some analysis is presented here. One run (named "bilat-pool-1") is as described above but is based on a single (non-randomized) fitness test per individual. Figure 2 shows the "trail" of 300 steps of the best individual after about 22000 fitness test.

Note that while this critter is capable of moving through "wide open spaces", it displays a pronounced preference to get close to a wall and move along it. When observed in real animals, this behavior is called *thigmotaxis*, and the term seems equally appropriate here. The critter seems to depart from closely following the outside wall only when it passes the vicinity of an obstacle on its other side. Evolution has discovered that there is a relatively uncluttered path around the edge of the world. (A feature which had not been intentionally designed in by the author.) This critter seems content to spend its life time circumnavigating the world. After some hand-simplifying of the control program, to removed redundant or ignored calculations, the circumnavigator is:

```
(turn (- 0.1
        0.02
        (% 0.02
          (look-for-obstacle 0.01))
        1
        (look-for-obstacle 0.1)
        (* (look-for-obstacle 0)
          (look-for-obstacle 0.01))
        0.1))
```

An interesting technical aspect of this program is that it does not contain any conditionals. The GP function set used here contains the conditional primitive `iflte` but in this case evolution chose to solve the problem without it. If a human programmer was asked to write a controller for this problem it is very likely that the resulting program would contain conditionals.

Because this individual avoided all collisions its raw score was 1.0, its final score was 0.64 because it got an unbalanced turn penalty of 0.93 (presumably because it was turning right as it circled the world), and it got an excessive turn penalty of 0.69, which is quite severe. The reason for this can be seen in the "gait" this creature developed. Looking at its trail one can see that it jitters from side to side on almost every step. This behavior has been observed in many of the evolved critters. One possible explanation of this is that since the critter has a limited set of sensors, it has many "blind spots". A gait that causes its sensors to sweep back and forth should provide it with much better sensor coverage of its path. A plot of best-of-run fitness and population-average fitness after every 1000 individuals is shown in Figure 3.

The second example (named "bilat-repeat-1") was identical except that its fitness function consisted of three separate runs through the Bilateral obstacle course with three slightly different starting positions. All three were within 1/3 of a body length of the original starting point. Before

the run there was some concern that these starting position were too similar and would not present the controller with enough variation to promote robust behavior. In fact, experience showed that most controllers got markedly different fitness values for each of the three starting conditions. The final fitness for each individual was defined to be the average of its three runs.

The best individual found appeared after about 31000 trials and had not been improved upon by 39000 trials. The fitness of that individual was 0.22, the average of its three runs which scored 0.46, 0.19, and 0.02. Its average penalties were 0.69 for excessive turning and 0.71 for unbalanced turning. The three overlaid "trails" for this individual are shown in Figure 4. A plot of best-of-run fitness and population-average fitness after every 1000 individuals is shown in Figure 5. The hand-simplified code for the best-of-run individual is:

```
(iflte (look-for-obstacle (look-for-obstacle 0.01))
     (* 2
       (look-for-obstacle 2)
       (look-for-obstacle -0.001))
     (turn (* (look-for-obstacle -0.01)
           (look-for-obstacle 0.1)))
     (+ (turn (+ (look-for-obstacle 0.1)
              (look-for-obstacle 0.1)))
       (turn 0.01)))
```

One odd feature of this run is that at one point the critter gets into a quasi-periodic looping behavior. Like spinning in place, looping is a valid solution to the fitness function as defined above, but it seems somewhat degenerate and does not fit our intuitive idea of "good" critter behavior. If we saw a real animal behaving like this we might assume it was distressed or ill. From a dynamic systems point of view, the semi-stable nature of this behavior is interesting: it loops for several cycles, but then suddenly escapes. This suggests a sensitive dependence on certain aspects of the environment. The first five (?) times around that loop the

controller decided to turn left, but the last time around, displaced only slightly from its state on the previous orbit, it decided to turn right.

Conclusions

The preliminary results reported here represent only partial solutions to the problem of robust vision-based collision avoidance. None of the behaviors evolved so far are robust, general purpose solutions to the full problem. So far the evolved behaviors seem to be either good but brittle, or mediocre but generalized. Thoughts about how to proceed beyond these limitations are given in the next section.

It seems significant that while a nice solution was obtained for the first example described above ("bilat-pool-1"), the second example ("bilat-repeat-1") seemed to converge before a good solution was found. Similarly, the first example attained a fitness value almost three times higher than the second example, even though the latter was allowed to run for almost twice as many individuals. This suggests that using multiple fitness cases makes the problem significantly more difficult to solve. It seems reasonable to conclude that multiple fitness tests require increased generality in the controller, and that generality tends to preclude the "easy" solutions which are opportunistic and brittle. The results presented here do not fully justify these conclusions, and more work is required to better understand these issues.

On the other hand, there is no question that vision-based collision avoidance strategies have begun to emerge, given only the requisite primitives, an appropriate fitness measure, and the action of Darwinian evolution through natural selection and survival of the fittest.

Future Work

Preliminary results from a more recent set of experiments similar to those presented here tend to confirm the notion that adding fitness cases increases the apparent difficulty of the problem, as can be

measured by the amount of "evolution work" (number of individuals processed) required to find a solution to the problem. Furthermore, attempting to add noise to the system as suggested in [Harvey 1993] seems to *significantly* increase the difficulty of the problem.

Future work in this area needs to investigate the effect of noise in controller programs. A systematic comparison should be made between the effect of using the average or the minimum of the multiple values obtained in fitness testing. This has been done in the neural net domain in [Harvey 1993] and we would like to compare those results to what happens in the GP domain.

As was noted above, spinning or looping behavior is in some sense degenerate and undesirable. To incorporate this bias, an alternative measure of fitness would be to keep track of how much area the critter has covered before collision. This could be accomplished, for example, by overlaying a grid on the critter's world and setting a flag in each grid cell the critter visits. The number of flagged cells would be a quantized approximation of area covered. Using this measure instead of, or in addition to, the number of steps taken before collision would promote collision-free paths that do not cross over themselves repeatedly. This "*exploratory* obstacle avoidance" criteria may yield a more interesting kind of behavior.

While the obstacles used in these experiments are static, nothing in the model depends on that fact. The critters have no maps of the world nor memory of any kind. As a result, their obstacle avoidance strategies are completely *reactive* [Brooks 1990] so we would expect them to be able to deal equally well with dynamic obstacles. An interesting variant of the current experiment would be to evolve avoidance in the presence of obstacles that move, or that appear and disappear "at random."

It is anticipated that better, more robust, obstacle avoidance strategies could be obtained by using a sequence of ever more challenging obstacle courses. But that approach puts a

significant burden on the human designer of the obstacles. First, there is the work involved in creating some large number of increasingly complex obstacle avoidance test cases. Second, as the critters become proficient at obstacle avoidance, it would become increasingly difficult to come up with obstacle courses that would challenge them. These considerations suggest that perhaps the obstacles should be seen as *predators* or *parasites* that coevolve with the critters. This kind of coevolution of problem-solvers and problem-poser has been examined in [Hillis 1990]. The "fitness" of an evolved obstacle would be judged by how well it could frustrate a critter's attempts at collision avoidance. In the confrontation between a given critter and obstacle, a collision early in the simulation would reflect poorly on the critter and well on the obstacle. Conversely, a collision late in the simulation (or the absence of a collision) reflects well on the critter and poorly on the obstacle.

Acknowledgments

The author wishes to thank his current employer, Electronic Arts for sponsoring the preparation of the final draft of this paper, and for sponsoring ongoing research on related topics. The following remarks refer to the original work performed in the summer of 1992:

The research reported here was pure amateur science. It not an officially sanctioned project of any corporation, university, or other funding agency. The individuals and companies listed below helped the author out of the goodness of their hearts and their interest in the subject matter. For generous grants of computer facilities, I am deeply indebted to Tom McMahon and Bart Gawboy of Information International Incorporated, Koichi Kobayashi of Nichimen Graphics Incorporated, and Richard Hollander of Video Image Associates. Special thanks go to Andrea Lackey whose workstation I repeatedly commandeered. Heartfelt thanks to Andy Kopra who cheerfully put up with my intrusions to his busy production schedule, and who helped

brainstorm some of these ideas back at ALife II. Thanks to: Larry Malone, Dave Dyer, Jay Sloat, Dave Aronson, Joseph Goldstone, Glen Neufeld, DJ, Antoine Durr, and whoever else I forgot. Thanks to Charles Taylor of UCLA for supplying the word "thigmotaxis." F inally, this work would not have happened without John Koza's trail-blazing work on the Genetic Programming paradigm. Many thanks to John and to James Rice for their helpful and encouraging comments during this project.

References

Harold Abelson and Andrea diSessa (1981) *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*, MIT Press, Cambridge, Massachusetts.

Susan Amkraut and Michael Girard (1989) Eurhythmy, in *SIGGRAPH Video Review ,,* Item 8*,* Issue 52, published by ACM-SIGGRAPH.

Randall D. Beer (1990) *Intelligence as Adaptive Behavior: an Experiment in Computation Neuroethology*, Academic Press, Boston, Massachusetts.

Randall D. Beer and John C. Gallagher (1992) Evolving Dynamical Neural Networks for Adaptive Behavior, in *Adaptive Behavior* 1:91-122.

Valentino Braitenberg (1984) *Vehicles: Experiments in Synthetic Psychology* , MIT Press, Cambridge, Massachusetts.

Rodney Brooks (1986) A Robust Layered Control System for a Mobile Robot, in *IEEE Journal of Robotics and Automation,* 2(1).

Rodney Brooks (1990) Elephants Don't Play Chess, in *Designing Autonomous Agents,* Pattie Maes, ed. pages 3-15.

Rodney A. Brooks (1991) Intelligence Without Representation, in *Artificial Intelligence* 47, pages 139-160.

Dave Cliff (1991a) Computational Neuroethology: A Provisional Manifesto, in *From Animals To Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB90)*, Meyer and Wilson editors, MIT Press, Cambridge, Massachusetts.

Dave Cliff (1991b) The Computational Hoverfly; a Study in Computational Neuroethology, in *From Animals To Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB90)*, Meyer and Wilson editors, MIT Press, Cambridge, Massachusetts.

Dave Cliff, Philip Husbands, and Inman Harvey (1993) Evolving Visually Guided Robots, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 374-383 .

Lawrence Davis, ed. (1991) *Handbook of Genetic Algorithms* Van Nostrand Reinhold.

Richard Dawkins (1986) *The Blind Watchmaker*, Harlow Logman.

John C. Gallagher and Randall D. Beer (1993) A Qualitative Dynamical Analysis of Evolved Locomotion Controllers, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts , pages 71 - 80.

Hugo de Garis (1990) Genetic Programming: Building artificial nervous systems using genetically programmed neural network modules, in Porter and Mooney editors,

Proceeding of the Seventh International Conference on Machine Learning, pages 132-139, Morgan Kaufmann.

Michael Girard and Susan Amkraut (1990) Eurhythmy: Concept and Process, *The Journal of Visualization and Computer Animation*, 1:15-17.

Inman Harvey, Philip Husbands, and Dave Cliff (1993) Issues in Evolutionary Robotics, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 364-373 .

W. Daniel Hillis (1990) Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure, in *Emergent Computation*, Stephanie Forrest editor (a special issue of *Physica D* 42) pages 228-234, MIT Press/North-Holland.

John Holland (1975) *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan.

John Holland (1992) Genetic Algorithms, in *Scientific American* , July 1992.

I. Horswill (1991) Proximity Detection Using a Spatial Filter Tuned in Three-Space, in *AAAI Fall Symposium on Sensory Aspects of Robotic Intelligence*, November, 1991.

John R. Koza (1989) Hierarchical Genetic Algorithms Operating on Populations of Computer Programs, in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, California.

John R. Koza (1992) *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, ISBN 0-262-11170-5, MIT Press, Cambridge, Massachusetts.

Ulrich Nehmzow, John Hallam, and Tim Smithers (1989) Really Useful Robots, in *Intelligent Autonomous Systems 2*, pages 284-293, ISBN 90-800410-1-7, Amsterdam.

Pattie Maes and Rodney Brooks (1990) Learning to Coordinate Behaviors, in *Proceedings of the Eighth Nation Conference on AI* (AAAI-90), pages 796-802.

Maja J. Mataric (1992) Minimizing Complexity in Controlling a Mobile Robot Population, *Proceedings, IEEE International Conference on Robotics and Automation*, pages 830-835.

Olivier Renault, Nadia Magnenat Thalmann, and Daniel Thalmann (1990) A Vision-Based Approach to Behavioral Animation, *The Journal of Visualization and Computer Animation*, 1:18-21.

Craig W. Reynolds (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in *Computer Graphics*, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.

Craig W. Reynolds (1988) Not Bumping Into Things, in the notes for the SIGGRAPH '88 course *Developments in Physically-Based Modeling*, pages G1-G13, published by ACM-SIGGRAPH.

Craig W. Reynolds (1993) An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion, in *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, Meyer, Roitblat and Wilson editors, MIT Press, Cambridge, Massachusetts, pages 384-392 .

Gary J. Ridsdale (1987) *The Director's Apprentice: Animating Figures in a Constrained Environment*, Ph.D. thesis, (see also *Technical Report TR-87-2*) Simon Fraser University.

Gary J. Ridsdale and Tom W. Calvert (1990) Animating Microworlds from Scripts and Relational Constraints, *Proceedings of Computer Animation '90*, Geneva.

Karl Sims (1991) Artificial Evolution for Computer Graphics, in *Computer Graphics*, 25(4) (SIGGRAPH '91 Conference Proceedings) pages 319-328.

Guy L. Steele Jr. (1990) *Common Lisp the Language*, second edition, ISBN 1-55558-042-4, Digital press.

Gilbert Syswerda (1989) Uniform Crossover in Genetic Algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2-9, Morgan Kaufmann Publishers.

Gilbert Syswerda (1991) A Study of Reproduction in Generational and Steady-State Genetic Algorithms, in *Foundations of Genetic Algorithms*, pages 94-101, Morgan Kaufmann Publishers.

Jeffery Ventrella (1990) *Walker* (video shown at ALife III), produced at Syracuse University.

Darrell Whitley (1989) The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best, in *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116-121, Morgan Kaufmann Publishers.

Larry Yaeger (1993) Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context, in *Artificial Life III*, forthcoming.

-