Evolving Effective Visual Tracking through Shaping

Simon Perkins
Los Alamos National Laboratory
Mail Stop D436, Los Alamos, NM 87545. USA.*
s.perkins@lanl.gov

Abstract

Shaping is a way in which a human designer can provide assistance to a learning system to enable it to solve problems that would otherwise defeat it. Results are presented showing that shaping can significantly improve the final performance of controllers evolved for a difficult visual tracking task. Controllers are developed in simulation and then transferred to a real robot head.

1 INTRODUCTION

1.1 ROBOT LEARNING FOR VISUAL TASKS

In recent years a lot of research effort has been put into producing real-world robot controllers automatically using techniques from the machine learning and optimization research fields. Engineering robot controllers by hand is difficult for many reasons, including problems of sensor and actuator noise, calibration errors, sensitivity of the robot-environment system to small changes in starting conditions, the requirement for real-time behaviour, and the sheer difficulty of humans 'thinking down' to the robot's level. The idea of deriving controllers for complex tasks automatically from the robot's own interaction with its environment is therefore an attractive one.

While the field has produced many interesting results, much work in robot learning is subject to the criticism that the tasks tackled are usually relatively simple. Typically the robots are equipped with only a few low-bandwidth sensors and actuators and are used in tasks such as obstacle avoidance and light-following which are robust to small inaccuracies in behaviour.

Equipping robots with vision opens up a whole field of more challenging tasks, and introduces a number of new problems to be overcome by a learning system. Visual sensors typically produce enormous amounts of raw data at a very high rate, and this data is generally difficult to relate unambiguously to the physical structure of the scene in front of the robot. A number of researchers have successfully used robot learning for visual tasks. Asada et al. (1996), for instance, have used Q-learning to train visually equipped mobile robots to shoot tennis balls through a 'goal'. Harvey et al. (1994) have evolved a neural network controller that allows a gantry robot to find coloured targets using vision. Finally, Jakobi (1998) has evolved a neural net motion-tracking controller for a 1 degree of freedom robot head.

1.2 LENDING A HAND: SHAPING

Any learning system has its limits. In general, the more flexible and general purpose the learning algorithm, and the fewer the assumptions made about the nature of the problem, the harder it is for the learning system to find a solution in a reasonable time. Many interesting visual tasks are probably beyond the capabilities of pure general-purpose learning algorithms without some kind of assistance — of the vision systems cited above, two (Asada et al., 1996; Jakobi, 1998) succeed by making use of heavily pre-processed visual input, which simplifies the problem enormously, while the third (Harvey et al., 1994) tackles a problem which it turns out can be solved by a relatively simple controller that looks at only two pixels in the image.

It seems clear that for really complicated tasks the learning system must be assisted by a human designer. In robot learning, giving assistance in this way is often called 'shaping' (Dorigo and Colombetti, 1993,

* Formerly at the Department of Artificial Intelligence, 5 Forrest Hill, University of Edinburgh, Scotland.

1998). There are several different categories of shaping method, but two of the most important are:

Controller decomposition Controllers for complex tasks can often be broken down into a hierarchy of smaller modules. It is often much easier to train these individual modules separately or sequentially, than to train the whole controller at once. In robotics, controllers are often decomposed in a behaviour-based way, with some modules performing simple sub-tasks, and others coordinating the activation of those modules (e.g. Mahadevan and Connell, 1992; Dorigo and Colombetti, 1993).

Progressive problem difficulty One important problem faced by learning robots is the difficulty of 'getting off the ground'. At the start of the learning process the robot will probably behave in a relatively random fashion, and in some training scenarios this might mean that the robot gets very little useful feedback on how to behave correctly. This problem can sometimes be alleviated by initially training the robot on easier versions of the full task. Asada et al. (1996) call this 'learning from easy missions'.

A fuller taxonomy of shaping methods can be found in Perkins (1999b); Perkins and Hayes (1996). For more discussion of robot shaping in general see Dorigo and Colombetti (1998).

2 VISUAL TRACKING

2.1 TASK DESCRIPTION

The experiments described here mostly concern a 1-D light-tracking task using a robot head equipped with a single camera that can be moved independently in 'pan' and 'tilt' directions. The task requires the controller to keep the camera pointed at a moving bright light source positioned in front of the robot, so that the straight-ahead direction of the camera is coincident with the centre of the light source. In these experiments, the robot head's tilt axis is fixed, and so only tracking errors in the horizontal direction are considered.

2.2 A SIMULATOR FOR EVOLVING REAL-WORLD TRACKERS

Evolving controllers for the tracking task would be a tedious process if carried out directly on the real robot. Although the camera can move quite rapidly, at around $100°s^{-1}$, the evolutionary runs presented in this paper would still take several weeks each to run. The robot head would be unlikely to survive this much continuous operation! As a result the controllers are initially evolved in simulation before being transferred to the real robot. Evolving in simulation has the very useful additional benefit that we can provide the learning system with a much more 'informed' evaluation mechanism than is possible in the real world.

Our simulator is designed using a methodology called the 'radical envelope of noise' (Jakobi, 1997), which can be summarized as:

- For environmental features that are *relevant* to the task, model them as well as possible, and add a small amount of noise to mask modelling inaccuracies.
- For environmental features that are *irrelevant* to the task, model them poorly, and add huge amounts of noise to prevent the learning system paying any attention to those features.

In accordance with this philosophy, a visual simulator for the light-tracking task was developed. For this task we are not really interested in the details of the background behind the target to be tracked, and so this can be replaced with a highly variable randomly generated one. Some care was taken to ensure that the simulated background shows random variation at both fine and coarse scales in a similar fashion to the real world. Figure 1 shows a typical simulated scene generated for the light-tracking task.

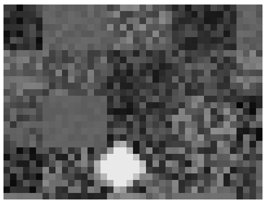

Figure 1: Example image produced by light-tracking simulator. Note that this image has been sub-sampled.

The image dimensions are 320×240 pixels. Background pixels vary in intensity value between 0 and 150. The diamond shaped simulated target varies in radius between $4°$ and $10°$, and in intensity between 230 and 240. The target moves in the simulated scene with a velocity between 0 and $10°s^{-1}$, and the simulator runs on a 4Hz cycle.

It is not necessary to model the background scene in a realistic way. However, it *is* necessary to model the way in which the target moves fairly accurately. This was achieved by basing the imaging model on empirical measurements of known scenes taken using the real robot head. Building a simulator using empirical data has previously been suggested by Miglino et al. (1995). On the actuator side, the stepper motors on the robot head actually allow almost error-free positioning, but the velocity signals sent to the simulator by the controller are corrupted by adding multiplicative Gaussian noise with $\mu = 1.0, \sigma = 0.02$ to mask image modelling inaccuracies. More details on the simulator can be found in Perkins (1999b).

3 EVOLUTIONARY DETAILS

3.1 THE TAG ARCHITECTURE

Many different learning architectures could be used for learning visual tasks. For reasons of generality, flexibility, and ease of use however, we use a variant of genetic programming (Koza, 1992) called TAG. The TAG architecture is described fully in Perkins (1999b), but a brief overview of its essential features is given here.

In contrast to standard GP, the structures evolved by TAG are general acyclic graphs rather than trees. Evolving graphs rather than trees allows controllers that use values computed by sub-graphs more than once to be be expressed in a more compact and evolvable way than is possible with simple trees. Several other authors have suggested adding graphs into GP for similar reasons (Poli, 1996; Teller and Veloso, 1996). TAG graphs are initially created by constructing a 'bag' of randomly chosen GP-style function and terminal nodes. The number of nodes in the bag is chosen randomly within a certain range: 10–20 nodes per bag in these experiments. TAG graphs can have more than one output value (for instance one output per actuator), and for each required output, a node is selected from the bag at random to return that value. If the node is not a terminal node, then further nodes are connected to its inputs as necessary and this connection process continues in recursive fashion until there are no nodes in the graph that need input connections. Nodes may connect to other nodes that already form part of the graph, but loops are prevented by insisting that no node may connect to a node that is its ancestor (i.e. its parent, or an ancestor of its parent). In order to ensure that the process terminates it is necessary that the bag contains at least one terminal node to start with. At the end of the connection process, not all nodes are necessarily in use — these unused nodes act as spare genetic material or 'introns' that may be used later.

Traditional GP uses sub-tree crossover as its primary genetic operator. TAG also uses crossover, but the operator must be modified for use with graphs. For the most part, 'sub-graph crossover' is very similar to sub-tree crossover. Starting with two parents, two offspring graphs are created by copying. Then, a node is selected in each offspring to act as an exchange node. To perform crossover, the exchange nodes are simply swapped between the two graphs, together with all their descendent nodes (a node is a descendent of another node if it is a child of that node or a descendent of a child).

The tricky part of sub-graph crossover is deciding what to do about connections that previously connected into the exchanged sub-graphs (other than connections to the exchange nodes themselves). One obvious answer is just to randomly reassign such connections, but TAG tries to take a more intelligent approach that attempts to preserve structure where possible. In brief, every node is associated with a fixed 'tag' value that is uniquely assigned when that node is first created. When a node is copied, the copy is given the same tag value. If a particular sub-graph turns out to be a useful component, then copies of it will tend to multiply in the population. Each of those copies will have similar sets of nodes and associated tags. When TAG is reassigning a connection into an exchanged region of nodes, it first checks to see if there are any nodes present in the newly formed individual that have the same tag as the node that the connection was previously connected to. If there is, then a connection is made to that node. If not, then the connection is reassigned randomly. The key idea is that it is less destructive if connections are reassigned to structures that are similar to the ones they were previously connected to.

Offspring can also be produced by mutation. In this case, a single child is initially generated by copying a single parent. Then, R nodes are selected at random for mutation, where R is Poisson-distributed with expected value 2.[1] Some types of node have internal parameters that are 'micro-mutatable', and if such a node is selected, then 90% of the time, one of its parameters is mutated slightly. In all other cases the node has one of its input connections reassigned randomly (if applicable), or is itself replaced with a random node.

TAG has a number of other interesting features, principally the use of a 'rational allocation of trials' (RAT)

[1]If $R = 0$, then R is re-generated.

mechanism (Teller and Andre, 1997) for reducing fitness evaluation time, but space precludes a fuller description here. See Perkins (1999b) for details.

Apart from the aforementioned exceptions, TAG is a relatively conventional evolutionary algorithm. It is a 'steady-state' algorithm in that as soon as offspring are generated they are put back into the evolving population — there is no concept of a generation. TAG maintains a population of size N. Parents are selected using tournament selection: at each evolutionary step, M individuals are chosen from the population at random and their fitnesses are compared. The fittest individual in the tournament is chosen as one parent, and one of the remaining individuals is chosen randomly as the other parent. 50% of the time the two parents are bred using crossover. Only one of the two potential children is actually generated. The other 50% of the time, mutation is used to derive a single offspring from the fitter of the two parents. In either case the offspring replaces the less fit of its parents. This evolutionary cycle is repeated a total of T times during a single run. In the experiments reported here, $N = 100$, $M = 4$ and usually, $T = 25000$.

3.2 SHAPING AND TAG

Shaping generally implies an incremental acquisition of behaviour with newly learned skills building upon previously acquired skills. The human trainer must design an incremental path of increasing competence that the robot is to follow. Each 'stage' along this path or 'shaping regime', results in the acquisition of another unit of competence in some area related to the overall task. The controller must not forget skills it has already learned, even if they are not directly relevant to the current stage of the training process, and it must be able to combine previously learned skills together to form new, more complex skills.

In these experiments, we achieve incremental learning using a simple multiple agent approach we call 'hierarchical evolutionary gating' (HEG). The idea is that the final controller consists of a collection of agents, each agent being evolved during a single stage of the shaping regime. Agents evolved at different stages may conflict with each other in the signals that they want to send to a particular actuator. In the HEG framework, such agents possess a special additional real-valued output called the 'validity'. For an agent to control an actuator, its validity must be greater than zero. If there is more than one such agent, then the most recently evolved agent wins. If no agents are eligible, the actuator assumes a default value for the current cycle. The validity mechanism allows an agent to take control if it needs to override previously evolved agents, and to relinquish control to previously evolved agents if that is not necessary. Note that since different agents can have different sets of outputs, an agent which is overridden with respect to actuator A, may get to control actuator B.

4 SHAPING EXPERIMENTS

4.1 SIMULATION RUNS

Two experiments were performed attempting to evolve controllers for the 1-D light-tracking task.

The first experiment acted as a control and attempted to evolve a monolithic controller consisting of a single TAG graph with a single output controlling the camera pan speed. Standard GP-style function nodes and terminals were used, consisting of the standard arithmetic functions: $+$, $-$, $\times$, $\div$, $>$, `neg`, `sgn` and `abs`; plus the ephemeral random constant $\Re$. $\div$ returns 1.0 if its second argument is zero. $>$ returns 1.0 if its first argument is greater than its second argument, or 0.0 otherwise. `sgn` returns 1.0 if its single argument is greater than zero, or -1.0 otherwise. Random constants are initialized to values between -1.0 and 1.0.

In addition to these familiar functions two additional terminal nodes and three additional functions were introduced for the tracking task. `vis` provides access to the camera sensor. It contains five internal parameters that define a rectangular 'receptive field' in the image. Four parameters define this rectangle, and the fifth defines a sampling resolution within that receptive field. `vis` is unusual in that it returns a 2-D *array* of values. The arithmetic functions described above are designed to cope with arrays in a sensible fashion — see Perkins (1999b) for details. In conjunction with this terminal node, three simple image processing function nodes are defined: `avg`, which returns the mean value of an array; `mx`, which returns the first x-moment of the array, and `my`, which returns the first y-moment of the array. These functions simply return their input if passed scalar values.

The fitness function for the light-tracking task is defined in terms of the tracking error E:

$$F = -\sum_{t}^{n}(1-\gamma^{t})E_t \qquad (1)$$

where E_t is the tracking error at time-step t, defined as the horizontal angular offset between the camera 'straight ahead' direction and the target direction, γ is a weighting constant, set to 0.5 in these experiments, and the trial is continued for n time-steps — in these experiments $n = 10$. Note that the sum is negated to ensure that higher fitness values are better.

A second experiment was performed to investigate whether shaping improved performance. The essential idea behind this shaping regime is to get the controller to first learn how to locate the target in the image, and only then to learn how to track the target. This is an example of controller decomposition. The first of these stages is broken down into four smaller stages, in which the controller has to locate a target appearing in successively larger and larger areas of the image — an example of progressive problem difficulty. For the light-location stages, the controller is required to signal its estimate of the position of the target by setting a 'memory unit' to a value representing the pan position of the target relative to the centre of the image. The fitness for these stages is then simply the difference between this estimate and the correct value, in degrees, negated so that higher fitness values are better. The same functions and terminals as used in the control experiment are available to all evolving agents. For the initial four light-location stages, each evolved agent possess an output targeting the memory unit, plus a validity output for conflict resolution. In the final light-tracking stage, the evolved agent has a single output controlling the pan speed, and an additional terminal node is made available that can read the value currently stored in the memory unit.

In detail, the five stage shaping regime looked like this:

1. 1-D light-location task with targets appearing only in the leftmost 25% of the image.

2. 1-D light-location task with targets appearing only in the leftmost 50% of the image. The first agent is frozen and a second evolving agent is added.

3. 1-D light-location task with targets appearing only in the leftmost 75% of the image. The second agent is frozen and a third evolving agent is added.

4. 1-D light-location task with targets appearing anywhere in the image. The third agent is frozen and a fourth evolving agent is added.

5. 1-D light-tracking task with targets appearing anywhere in the image. The fourth agent is frozen and a fifth agent is added. The new agent has access to a new terminal node `mem` which reads the memory unit written to by the first four agents.

Table 1: Comparison of end-of-run results for the light-tracking experiments. (a) Control experiment. (b) Shaped experiment. All tracking errors are given in degrees.

	Min. Tracking Error			Best Individual	
	10%	Median	90%	Error	Hits
(a)	6.0	7.9 – 9.0	10.3	5.7 ± 0.4	95.2%
(b)	4.3	5.5 – 7.2	9.8	4.2 ± 0.5	97.3%

Each stage is 25000 tournaments long. The control experiment was continued for 125000 tournaments to provide a fair comparison. Each experiment was repeated 50 times with different random number seeds to get good statistics.

Table 1 shows the end-of-run performances from the two experiments. The first three columns show how final performance varied over the 50 runs, and show the median, and 10th and 90th percentile tracking errors of the best individual at the end of each run. The range shown for the median gives the 95% confidence interval. A two-tailed randomization test (Cohen, 1995) showed that the controllers produced by the shaped experiment have a significantly lower median error than those produced by the control ($p < 0.01$). The last three columns show the tracking performances of the best individuals seen from each experiment. The hit rate indicates what percentage of the time the controller kept the target in view for a whole trial.

4.2 BACK TO REALITY

As mentioned at the beginning of this paper, it is one things to show that shaping can help us produce better robot controllers in simulation, but another thing altogether to show that this leads to better controllers in reality. To test the ability of the evolved controllers to work on the real robot, further shaping runs were carried out to attempt to evolve controllers for the light tracking task that could compete with hand-designed controllers. Space precludes a detailed description of the experimental procedure here, but a full description can be found in Perkins (1999b). In brief though, a ten stage shaping regime, based on the one investigated above, was used to produce a controller that performed well on the full pan and tilt light-tracking task in simulation. This controller was then transferred directly to the real robot head upon which the simulator was based, and tested in the manner described below. A handmade controller, using a simple 'threshold and centroid' technique to decide how to move the camera, was also produced, and this was tested in the same way.

The light-tracking controllers were tested by darkening the room, and then holding a desk lamp in front of the robot at a distance of about 1.5m from the camera. The lamp was held stationary in 100 different positions, and the controller was allowed to attempt to point the camera at the light. Performance was measured by using simple image processing to locate the centre of the light in the image, and determining the offset from the optical axis of the camera.

Table 2 shows the results from the real-world evaluations.

Table 2: Comparison of real-world performances of evolved and hand-made light-tracking controllers.

Controller	Mean Error
Evolved light-tracker	5.09 ± 1.49
Hand-made light-tracker	1.41 ± 0.15

Subjectively, the evolved controller tracked the light as well as the hand-made controller. The objective comparison above suggests that the hand-made controller does better, but this is misleading since the algorithm used to locate the light for the purposes of evaluation is the same as the one the hand-made controller uses to track the light. Hence a good score is guaranteed!

5 CONCLUSIONS

The shaping experiments presented here, along with additional experiments presented in Perkins (1999b), show that if used carefully, shaping can significantly improve the final quality of controllers evolved for complex tasks. In Perkins (1999a), similar techniques were used to evolve a controller that could track arbitrarily coloured moving objects with both pan and tilt axes. Whereas controller decomposition based around 'control-flow' has been used several times before to improve learning performance (e.g. Dorigo and Colombetti, 1993), the 'data-flow' decomposition used here is, I believe, novel.

The paper also provides a further demonstration of the process of transferring a controller that has been evolved in simulation onto a real robot, using a complex visual task.

References

Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1996). Purposive behaviour acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303.

Cohen, P. (1995). *Empirical Methods for Artificial Intelligence.* MIT Press.

Dorigo, M. and Colombetti, M. (1993). Robot shaping: Developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkley, CA 94704.

Dorigo, M. and Colombetti, M. (1998). *Robot Shaping: An Experiment in Behaviour Engineering.* MIT Press/Bradford Books.

Harvey, I., Husbands, P., and Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In Cliff, D., Meyer, J.-A., and Wilson, S., editors, *From Animals to Animats 3: Proc. 3rd Int. Conf. Simulation of Adaptive Behavior.* MIT Press.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6:325–368.

Jakobi, N. (1998). Evolving motion-tracking behaviour for a panning camera head. In Pfeifer, R., Blumberg, B., Meyer, J.-A., and Wilson, S., editors, *From Animals to Animats 5: Proc. 5th Int. Conf. Simulation of Adaptive Behavior.* MIT Press.

Koza, J. R. (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behaviour-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365.

Miglino, O., Lund, H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artifical Life*, 2(4):417–434.

Perkins, S. (1999a). Evolving robot vision: Increasing performance through shaping. In *Proc. 1999 Congress of Evolutionary Computation.* (In press).

Perkins, S. (1999b). *Incremental Acquisition of Complex Visual Behaviour using Genetic Programming and Robot Shaping.* PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 5 Forrest Hill, Edinburgh, Scotland. Available from `http://www.dai.ed.ac.uk/students/simonpe/pubs.html`.

Perkins, S. and Hayes, G. (1996). Robot shaping — principles, methods and architectures. Technical Report 795, Dept. of Artificial Intelligence, Univ. of Edinburgh, Scotland.

Poli, R. (1996). Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, UK.

Teller, A. and Andre, D. (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In Koza et al., editor, *Proc. Genetic Programming '97.* Morgan Kaufmann.

Teller, A. and Veloso, M. (1996). PADO: Learning tree-structured algorithms for orchestration into an object recognition system. Technical report, Department of Computer Science, CMU, Pittsburgh, PA.