Tracking Object Positions in Real-time Video using Genetic Programming

W. Smart and M. Zhang

School of Mathematics, Statistics and Computer Science,
Victoria University of Wellington,
P. O. Box 600, Wellington, New Zealand
{smartwill, mengjie}@mcs.vuw.ac.nz

Abstract

This paper describes a new approach to the use of Genetic Programming (GP) to evolve programs for tracking objects quickly in streaming video. A small number of images, with located objects, are used as training data and GP automatically performs feature-selection on these images at the pixel level. The use of feature functions is introduced, taking a single offset argument, in contrast to the standard feature terminal approach. The features include both "directionless" intensity features and "directional" edge detection features. The fitness function rewards evolved programs that can move training points, located on a grid around an object, closer to the object. As such, a good program will also be able to update an object position from frame to frame for tracking. Two video sequences are examined, with evolved programs tracking the left-eye and forehead of a person successfully. The method is very fast, tracking a frame in six or seven milliseconds on a 2.6GHz PC.

Keywords: Artificial Intelligence, Genetic Programming, Object Tracking, Computer Vision

1 Introduction

Many object tracking tasks arise in computer vision, whether it be for surveillance [1], video compression, novel human-to-computer interfaces [2] or human head tracking [3, 4, 5]. A wide variety of innovative methods have been used to solve this problem. However, many of these methods are either very specialized to the type of object that is being tracked [3, 4] or very complex, requiring powerful computers or specialist hardware to run in real-time [2, 6, 5]. In this paper we present a very fast method of tracking objects which makes no assumptions about the properties, such as shape, of the object to be tracked. Rather, a genetic search system is used to find such knowledge, given only a few training images with known object locations.

Genetic Programming (GP) [7, 8] is a fast developing search method with origins in Genetic Algorithms (GAs) and Automatic Programming. In GP, as in GAs, a population of individuals (called evolved programs in GP) is created, then evolved stepwise from generation to generation, using a fitness heuristic. From one generation to the next, the individuals in the population are updated by genetic operators which aim to keep material from, swap material between, and add random material to selected individuals. As individuals that have better fitness are more likely to be selected to be updated by the genetic operators, the method has parallels with individuals in species' populations and Darwinian natural selection.

The main goal of this paper is to develop a method to evolve programs using GP that can successfully track objects from frame to frame in a video sequence, given the frame images, and the position of the object in the first frame of the sequence. The training system will not require user-specified object properties, but only a small number of training images with objects at known locations.

This paper is organized as follows: In section 2 the new primitive set and feature representation are presented. In section 3 the fitness function is presented. In section 4 the tracking tasks are given, and results are shown. In section 5 the conclusions and future directions are presented.

2 Evolved Program Representation

In this paper we use the widely-used representation for evolved programs, the Lisp S-expression [7]. In this representation, the evolved programs can be represented as trees, with functions at the internal nodes, and terminals at the leaf nodes.

The information available to the tracking system at any frame in the video-sequence is the frame image, and the position of the tracked object in the

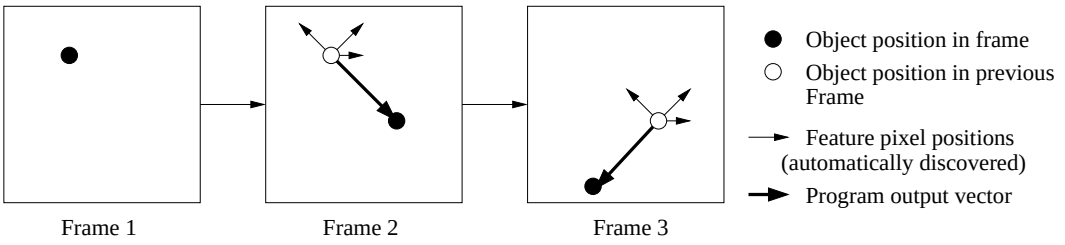


Figure 1: Tracking of an object position by adding output vectors to previous positions.

previous frame. In this approach, we find a vector (called the update vector) that can be added to the position of the object in the previous frame, moving it closer to the position in the present frame. The method is shown in Figure 1.

In frame one, the position is initialized using prior knowledge or a localization method. In frame two and three, the evolved program used as a tracker is evaluated using as features the values of pixels situated around the previous position of the object (outline circle). The evolved program outputs the update vector, which is used to produce an estimate of the object position in the current frame.

To do this, instead of the standard approach where programs return a single real value, programs in our representation return a two-valued vector, containing x and y components. This vector may be directly added to the previous position in order to move it closer to the present object. The argument types and return types of all terminals and functions used in the system are therefore two-valued vectors.

2.1 Terminal Set

In this approach, there is only one terminal type: numeric terminals. Numeric terminals are randomly generated two-valued vector constants. Both the x and y components are randomly generated from a standard normal distribution.

A major difference from the standard approach is the use of a feature function, instead of the usual use of a feature terminal. In standard GP the features used as input from the environment come in fixed-size feature vectors. The vector values are included in the evolved program as terminals, each returning a specific feature from the vector.

2.2 Function Set

The features of evolved programs in our system are functions taking one vector argument which specifies the position in the image to use for the feature. The exact position is the value of the vector argument added to the current estimate of object position, or the position that the program is "evaluated at". There are two feature function types, directionless and directional. The "directionless feature" function returns the pixel intensity in the x component, and an omni-directional edge detection value at the pixel location in the y component. The "directional feature" function returns an horizontal edge detection value in the x component, and a vertical edge detection value in the y component.

The functions used in experiments are listed In Table 1.

In Table 1, a_i is the i'th argument passed to the function, which will have x and y components ($a_i.x$ and $a_i.y$). $I(v)$ is the intensity of the pixel in the current frame, at a position of v relative to the current best estimate of the object position. $E_x(v)$ is $I(v + \{3, 0\}) - I(v - \{3, 0\})$ and $E_y(v)$ is $I(v + \{0, 3\}) - I(v - \{0, 3\})$.

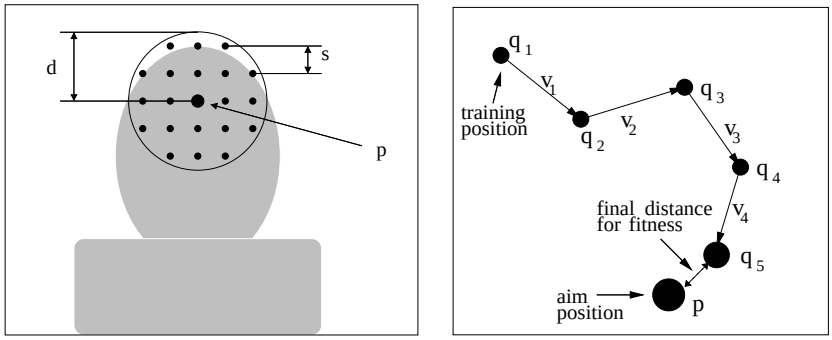
The functions satisfy *closure*, that is all functions can cope with all possible argument values. Also, the functions aim to satisfy *sufficiency*, that is, there exist programs using these functions that can solve the task. In order to approach this, standard vector mathematics is included, such as addition, subtraction, dot product (multiplication), cross product. Additionally, discontinuities are allowed by a conditional function (if) and rotation about the origin by any angle is also included.

3 Fitness Function and Training

In this approach an evolved program will output a vector when evaluated at a position in an image. When added to the position, the vector output by a good program, will refine the position to be closer to the true position of the object. As such the training examples consist of a grid of positions that are *near* the object (of known location) in an image. These training positions are set at the start of evolution, and do not change from one program to the next. The goal of an evolved program is

Table 1: The functions used in experiments.

Function	resultant x	resultant y
Addition	$a_1.x + a_2.x$	$a_1.y + a_2.y$
Subtraction	$a_1.x - a_2.x$	$a_1.y - a_2.y$
Multiplication	$a_1.x \times a_2.x$	$a_1.y \times a_2.y$
Protected Division	$\frac{a_1.x}{a_2.x}$ or 0 if $a_2.x = 0$	$\frac{a_1.y}{a_2.y}$ or 0 if $a_2.y = 0$
If	$a_2.x$ if $a_1.x < 0$ else $a_3.x$	$a_2.y$ if $a_1.y < 0$ else $a_3.y$
Determinant	$a_1.x \times a_2.y - a_1.y \times a_2.x$	$a_1.y \times a_2.x - a_1.x \times a_2.y$
Rotate by $a_2.x$	$a_1.x \cos(a_2.x) + a_1.y \sin(a_2.x)$	$a_1.y \cos(a_2.x) - a_1.x \sin(a_2.x)$
Rotate by 90 / swap	$-a_1.y$	$a_1.x$
Combine a_1, a_2	$a_1.x$	$a_2.y$
Directionless Feature	$I(a_1)$	$\sqrt{(E_x(a_1))^2 + (E_y(a_1))^2}$
Directional Feature	$E_x(a_1)$	$E_y(a_1)$



(a) (b)

Figure 2: (a) A simplified drawing of a person's head, with training positions arranged as a grid around the target point p at the centre of the forehead. (b) The iteration process by which the evolved program may refine its position estimate.

to, starting from each training position, move to a point that is closer to the surrounded object.

Figure 2(a) shows a simplified image, with training positions occupying points on a grid, with spacing s, and a maximum of distance d from the desired object position p for the image.

3.1 Fitness Function

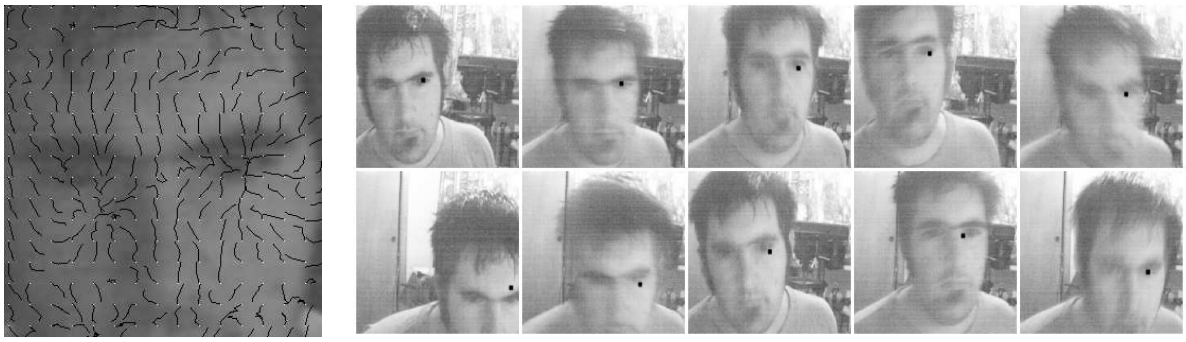
In the case of Figure 2(a), the aim of the evolved program is to move a given point closer to the centre of the forehead (point p). The vector output by the evolved program is added to the given point, giving a new point estimate which is, hopefully, closer to p. However, the programs evolved can be somewhat roundabout in their attempts to get to p, and this is allowed so long as the program moves the training points close to p, after a few iterations. This is shown diagrammatically in Figure 2(b).

The first point used is the training point, which we call q_1. When the program is evaluated at point q_i, it produces a vector o_i. If the length of o_i is less than a maximum length l, then it is used directly. Otherwise o_i is shortened to length l without changing its direction. The (possibly shortened) o_i vector is called v_i. An updated position is found as $q_{i+1} = q_i + v_i$. The program is then evaluated at position q_{i+1}, producing v_{i+1} and then q_{i+2} by the same equation used for q_{i+1}. This process is repeated a set number of times for each training point, finally producing the point r (q_5 in Figure 2(b))

The fitness function is shown in Equation 1, where f is the fitness, p is the position of the desired object, r_j is the refined position of the j'th training position around p. N is the number of training positions around the object in the image.

$$f = \sum_{j=1}^{N} \sqrt{(p.x - r_j.x)^2 + (p.y - r_j.y)^2} \quad (1)$$

It can be seen that Equation 1 will be lower for programs that refine the training points to be closer to p, while being higher for programs that refine the training points to be further away.



(a) (b)

Figure 3: Tracking by evolved program trained to follow the left eye. (a) Paths taken by the program on a sample face image. (b) Every 20th frame in the first video-sequence, showing tracked position as black dot.

4 Experiments and Results

Experiments were run to evaluate how well this approach can track a human head in low bandwidth, greyscale video sequences. In this section we will discuss both the experiment setup and the results gained using this approach.

4.1 Datasets

The video sequences were recorded to AVI format files using a standard webcam in greyscale at 352×288 resolution. Due to the use of a webcam, the quality of the video is not very good, fast motion causes a very blurry image. This makes the job of a tracker much harder due to the low quality of the still images it must work with.

Two sequences were used, each with a person's head moving at a fixed distance from the webcam. In the first sequence the head moves relatively slowly and there is no occlusion. The task here is to track the left-eye of the person. In the second sequence, the head moves quickly at times, and is occluded at the end. The task here is to track the forehead of the person. Clearly, the tracking task in the second sequence is harder than in the first.

4.2 Experiment Configuration

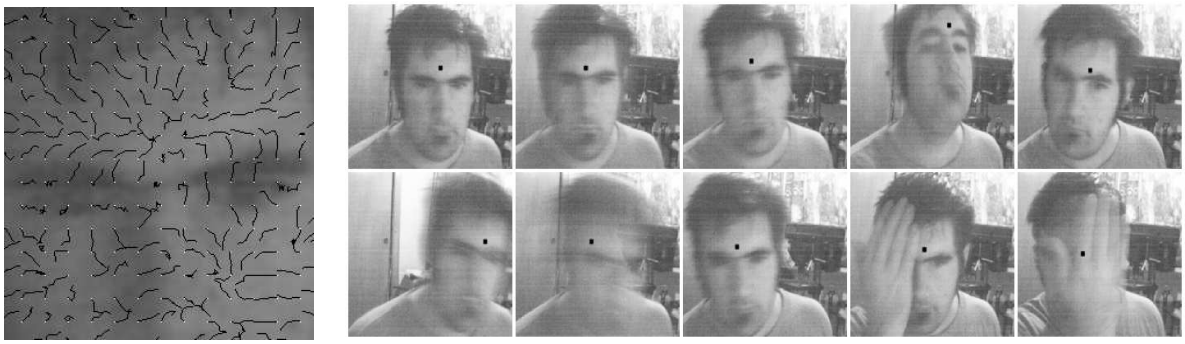
For evolution, the population size was 500, the number of generations was 150, the maximum depth of the evolved programs was eight. The initial population of programs was generated using the ramped half-half method [7] with an initial maximum depth of six. In each population 10% came from reproduction, 30% came from mutation and 60% from crossover. A 3-way tournament selection mechanism was used.

All programs were trained on the same images, 15 frames from the second sequence selected for their range of content (such as pose and motion blur). The only differences in training between programs that track different objects were the positions marked as the objects in the images. The grid parameters were $<s = 8, d = 40>$ for the left-eye-tracker program, and $<s = 10, d = 60>$ for the forehead-tracker program. For the tracking output, the value of l was 2.0 pixels, and the number of iterations of the position update process per frame was 80. The computer used for processing was a Pentium IV 2.6GHz PC.

4.3 Experiments

Figures 3.1 and 4 show results of programs evolved to track the left-eye and forehead, respectively, in the two video-sequences. Figures 3.1(a) and 4(a) give details of a sample face image, showing only the centre of the face. The white dots in the figures show points from which the program was started, and the black trails show the path by which the program follows as it refines this position for a few iterations. Figures 3.1(b) and 4(b) show every 20th frame in the first and second video sequences respectively, with a black dot in every frame indicating the tracked position of the object by the evolved program. The frames go from top-left to top-right, then bottom-left to bottom-right.

In Figure 3.1(a) the paths that are close to the left eye are seen to generally move toward the centre of the eye. In Figure 3.1(b) we can see that the tracked position closely follows the path of the left eye. In the bottom-left frame we can see that the tracked position is slightly off the true position. This is due to the eye being very near the edge of the frame. However, as the eye moves back into the field of view, the tracking position realigns



(a) (b)

Figure 4: Tracking by evolved program trained to follow the centre of the forehead. (a) Paths taken by the program on a sample face image. (b) Every 20th frame in the second video-sequence, showing tracked position as black dot.

itself. This suggests that this evolved program had stong adaptability even when part of the eye was occluded by the image border. Some other evolved programs would get attracted to the edge of the image at this part of the sequence. The tracking process for each frame in this sequence took about six milliseconds.

In Figure 4(a) the paths that are close to the forehead are seen to generally move inward, toward the centre of the forehead. In Figure 4(b) we can see that the black dot closely follows the path of the forehead, even when the head is rotated in the fourth frame and moved quickly in frames six to eight. In the ninth and tenth shown frames in Figure 4(b), we see that the tracked point is still correct, even when the face was completely occluded by the hand. This is a property of the evolved program used for tracking. Some other evolved programs would follow the hand, or move erratically when faced with this occlusion. Similarly, all evolved programs tested were different in terms of maximum tracking speed, jitter, tolerance to head pose and other properties. The tracking process for each frame in this sequence took about seven milliseconds.

5 Conclusions

The goal of this research was to develop a method to evolve programs, using GP, that can track the position of an object in a video sequence. This goal was successfully achieved by the introduction of a two-value vector program representation and the use of feature functions in the system.

In this approach, each evolved program returns a two-valued vector, rather that only a single floating-point number as in standard GP. The evolved programs do not use the standard method of feature terminals, but rather feature functions which take a single vector as an argument. The position in the image that the feature describes is the value of the vector argument, added to the position that the evolved program is "evaluated at".

When evaluated at a starting position that is close to an object in an image, an evolved program that is trained to track the type of object returns a vector indicating the direction and distance to move from the starting position in order to get closer to the object. The fitness function uses the sum of distances from the true object position to the refined positions of a set of training positions near the object.

In experiments using low-quality, greyscale video sequences, the evolved programs did track the objects in the image very successfully. The evolved programs were trained to either follow the centre of the forehead, or the left eye. The only difference in training for these very different objects was the positions given for the objects in training images, yet each of the evolved programs did track its object successfully. The actual tracking process was found to be very fast, which would use about 10% of the power of a 2.6GHz PC if it were used for real-time video. With optimization of the code this could be lowered even further.

Note that each evolved program produced did differ in abilities such as speed, jitter and the range from the object that could be successfully tracked. As such, when applying this technique, it is important to choose an evolved program that has the desired tracking properties.

For future work, we will investigate the use of more than one evolved program for tracking an object in video-sequences, with strengths combined from the

different programs. Each would track either the same object or different ones (such as one each for the left eye, right eye and centre of the forehead). The method will also be compared to other tracking methods, and strengths and weaknesses of the methods assessed.

References

[1] L. M. Fuentes and S. A. Velastin, "People tracking in surveillance applications," 2001.

[2] C. R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780–785, 1997.

[3] S. Birchfield, "Elliptical head tracking using intensity gradients and color histograms," 1998.

[4] P. Fieguth and D. Terzopoulos, "Color based tracking of heads and other mobile objects at video frame rates," 1997.

[5] J. Denzler and H. Niemann, "Evaluating the performance of active contour models for real–time object tracking," in *Second Asian Conference on Computer Vision*, vol. 2, (Singapore), pp. II/341–II/345, 1995.

[6] J. Denzler and D. W. R. Paulus, "Active motion detection and object tracking," in *ICIP (3)*, pp. 635–639, 1994.

[7] J. R. Koza, *Genetic Programming: on the programming of computers by means of natural selection*. London, England: Cambridge, Mass. : MIT Press, 1994.

[8] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers, 1998.