Image Tracking in Real-Time: a Transputer Emulation of some Early Mammalian Vision Processes

P.H.Welch and D.C.Wood
Computing Laboratory, University of Kent at Canterbury,
Canterbury, Kent, England

Abstract

By emulating some of the 'early' vision processes believed to occur in the visual cortex of mammals, dramatic savings can be made in the computational efforts usually associated with image processing. This paper describes the design and implementation of a low-cost visual tracking system that uses no special signal processing, vector processing nor floating-point hardware. Its effectiveness relies only on the parallel replication of fast scaler integer processors with low latency communications. The current implementation uses 10 T425 transputers, processes 544 × 544 byte-pixel images at camera frame rates and provides 100 Hz. tracking feedback signals to the camera pan-and-tilt motors with a latency of no more than one-fifth of a second. The system will track any object that occupies the (majority of the) centre of the field of view moving against any background — no special lighting conditions or artificial scenery are required. The design and implementation follow naturally parallel structures and are expressed at all levels in *occam*.

1 Introduction

Biological systems have evolved highly efficient mechanisms for the accurate tracking of moving objects. Rather than wielding a computational sledge-hammer, it is a good idea for computer systems designers — charged with solving the same problem — to look at nature and see what can be 'borrowed'.

The algorithms and data-structures employed within KITTEN (the Kent Intelligent Target Tracker) exploit the following features of mammalian vision:-

- *the fovea:* light receptors are concentrated at the centre of the retina;
- *adaptation:* an individual receptor quickly stops firing if the light level falling upon it remains constant;
- *rapid eye movement:* the eye never remains still — there are always 'tremors', even when apparently at rest;
- *foveal tracking:* only objects maintained in the centre of the field-of-view are tracked — we cannot track objects using peripheral vision;
- *velocity modelling:* there is some physiological evidence [1] that suggests that eye movement during object tracking is controlled by modelling eye velocities rather than absolute eye position.

BMVC 1992 doi:10.5244/C.6.40

The differing densities of retinal light receptors from the fovea (the highest) to the periphery (the lowest) allow the same scene to be processed with a range of views and resolutions. Each view — from the narrow angled, but highly resolved, foveal patch to the wide angled, but low resolution, full scene — demands a similar quantity of memory and processing power. Each view offers differing, but complementary, information to higher level tasks (such as tracking). All these views can be processed in parallel.

The adaptation of retinal light receptors to constant stimuli mean that they are naturally sensitive to moving objects — this allows moving objects to be detected and 'acquired' (i.e. the eye is moved to locate the detected object on its fovea). During tracking, this same effect enables objects that change their (angular) velocities, with respect to the eye, to be detected and for correcting signals to be sent to the eye muscles.

The rapid movement (or 'dithering') of an apparently resting eye continuously shakes the image falling on its retina. If this did not happen, the adaptation effects. would cause a stationary eye to become blind. The effects of adaptation and dithering need to be well-balanced!

The eye can only track objects if they can be kept 'locked' in the fovea. During tracking, the tracked objects will appear to be stationary, but the background will be moving. If the tracking response mechanisms for the eye operated across the whole scene, correction signals generated by the moving background (usually the majority of the scene) would quickly bring the eye to a halt. Tracking would only be possible on objects that occupied the majority of the whole scene or on smaller objects moving across a uniform background. Clearly, the eye does much better than this!

Finally, there is some debate as to whether a tracking eye is controlled by sending it change of velocity information (i.e. object accelerations) or change of position information (i.e. object velocities). The latter would imply that the eye needs to be repeatedly brought to rest momentarily, whilst the continuing velocity of the tracked object was observed. The former implies that observations can continue 'on-the-fly'. Because the motors in the pan-and-tilt unit on which our camera is mounted have considerably greater inertia than the muscles that control eyeball (or head) movement, we choose the former model. It also seems more simple and elegant.

2 Top-Level Design

Figure 1 shows the basic feedback control loop for the tracker. Images acquired from the camera are processed into control signals for operating the pan-and-tilt motors on which the camera is mounted. Objects moving in the centre of the field of view cause the camera to move to try and follow it — this, of course, impacts back upon the image acquired. Clearly, the lower the time around this feedback loop, the better will be the quality of the tracking response.

During tracking, the movements of the target and the angular velocities of the camera pan-and-tilt unit need to be reasonably matched. Any discrepancies — due to errors in processing or real motion changes by the target — cause deviation of the target from the centre of the acquired image. These deviations must be measured in real-time and correction signals returned to the pan-and-tilt motors to bring the target image back on track.

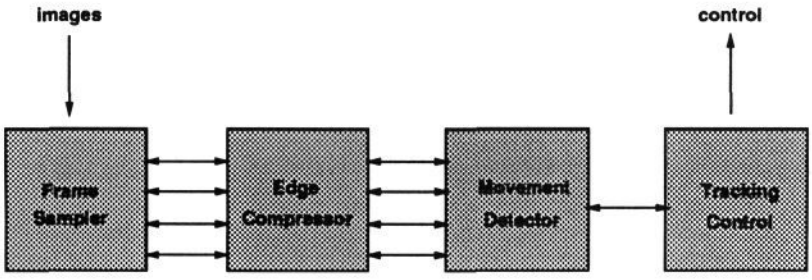


Figure 1: Image Processing Pipeline

Figure 1 also shows the top-level processing structure — a four-stage pipeline. Currently, the `Frame Sampler` and `Tracking Control` use one transputer each, whilst the `Edge Compressor` and `Movement Detector` both use four. The `images` input represents frames generated by the camera at 25 cycles per second. The `control` outputs are digital/analogue signals for operating the pan-and-tilt motors — these are generated at 100 cycles per second. The internal channels represent *transputer* links.

Information flows continuously from `images` through to `control` feedback. Occasional control message flow in the reverse direction — these are to change the mode of operation of certain processes and for the fine-tuning of various parameters.

3 Frame Sampling and Dithering

Our *transputer*-controlled frame-grabber delivers one 544 × 544 8-bit greyscale image into video-RAM every 40 milli-seconds (i.e. 25 frames/second). This represents a data bandwidth of about 7 Mbytes/second, whereas the capacity of the four links leaving this transputer (and routed, in our prototype system, through electronic switches) is only about 5 Mbytes/second.

Each frame-grabbed 544 × 544 image, therefore, is reduced to four 68 × 68 images called `layer[0]` through to `layer[3]`:-

- `layer[0]` is just the middle 68 × 68 square of the original image. It corresponds to the 'foveal patch' on the retina, where image sampling density is greatest;

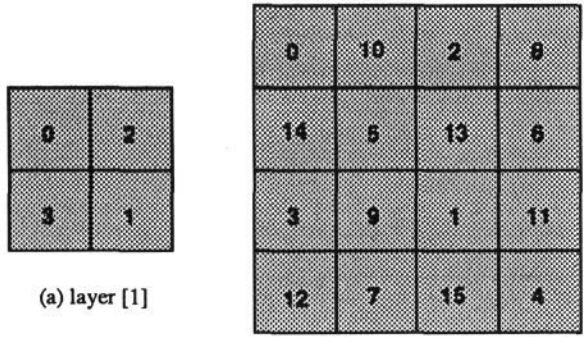
- `layer[1]` is the middle 136 × 136 square of the original image sampled every other pixel in each dimension — i.e. from each 2 × 2 square in this patch, one pixel value is chosen;

- `layer[2]` is the middle 272 × 272 square sampled every fourth pixel — i.e. from each 4 × 4 square, one pixel is selected;

- `layer[3]` is the original 544 × 544 image sampled every eighth pixel. This corresponds to the 'widest angle' view of the scene, but has the lowest resolution.

This sampling pattern was previously described in [2], where it was used to provide input to the 'Self-Similar Stack' — see also [3, 4, 5, 6]. Using the language from this stack model of vision, we shall refer to `layer[0]` as the 'top' and to `layer[3]` as the 'bottom'.

Rapid eye movement (or dithering of the sampled image) is a by-product of the pixel sampling algorithm used. On `layer[1]`, `layer[2]` and `layer[3]`, when pixels are selected from each (respectively 2×2, 4×4 and 8×8) sub-square, a different location is chosen in successive frames. The dithering sequence is not a straight row-by-row scan of each sub-square. Instead, the sequence jumps around the square, choosing for each successive location a pixel from an area not recently visited. On `layer[1]`, the sampling sequence for each 2×2 sub-square (of the 136×136 inner square) is given by Figure 2(a):-



(a) layer [1]

(b) layer [2]

Figure 2: Dithered Sampling Sequences

The `layer[1]` sequence is used recursively to generate the rather longer sequence for `layer[2]` (Figure 2(b)) and `layer[3]` (not shown). The aim of this dithering is to ensure that a 'good' representation of each sub-square is chosen over the last n samplings — for any value of n.

Viewing the result of this dithered sampling in real-time (i.e. 25 frames/sec.), the images from the lower layers (i.e. `layer[1]`, `layer[2]` and `layer[3]`) show gradually increasing amounts of 'tremor'. These tremoring effects are very useful for the next stage in processing — edge extraction.

The reduced images from each layer are output down the corresponding link. At 25 images/second, each link carries approximately 100 Kbytes/second — well within its capacity.

4 Edge Compression

Before movement detection is attempted, the images are reduced still further by the removal of texture information. This edge-filtering operation needs to be computationally light in order to cycle at camera frame rates.

In [5, 6], the implementation of algorithms based on the Self-Similar Stack, Hierarchical Discrete Correlation and Difference of Gaussians [2, 3, 4] are described. Using a pipe-line of 3 T800 transputers per level (i.e. floating-point arithmetic is necessary), the necessary computational bandwidth can be

achieved — but at some cost in latency (about 200 milli-seconds for just this stage).

In KITTEN, a much lighter approach is used. Instead of trying to extract edges from each image separately, we make use of the fact that they form part of a moving sequence. Following (loosely) the adaptation characteristics of retinal light receptors, successive images are simply differenced (absolutely) and then thresholded. Parts of the image that have not changed simply disappear. Objects that have *moved* (slightly) are reduced to just their edges with varying levels of thickness (from greatest along the axis perpendicular to the direction of movement, down to zero along the axis parallel to the direction of movement). These filtering effects work well for natural objects regardless of their orientation or direction of movement — i.e. we are not restricted to just the major points of the compass.

Because of the dithered image sampling, all *stationary* images in the bottom three layers have an apparent tremor. This is sufficient for the differencing computations to yield their edges. If the system were perfectly locked on to a (constant velocity) *moving* object, its image would be stationary — this way, we can still see it!

Images from each layer are processed in parallel by separate *transputers*. The thresholding level that determines whether the difference between sampled pixel values (in consecutive frames) are sufficiently different may be adjusted dynamically by (backwards flowing) control signals — these are 'higher level' decisions.

After thresholding, the 68×68 byte-pixel images have only binary values — 'edge' or 'not-an-edge'. These are now compressed into 68×68 bit-pixel images. Each row is squashed on to three (32-bit) integers with a generous overlap.

The output image from each layer has now been reduced to just 816 bytes, lowering the data-flow on each link to only 20 Kbytes/second.

5 Movement Detection

The method for extracting movement information from the sequence of images is the same as that reported in [2, 6]. However, with the bit-compressed representation of edge-only images now available, the required computational effort is more than one order of magnitude less.

For tracking purposes, movement detection should only be sought within the 'foveal patch' of each image. Four streams of images, `layer[0]` through `layer[3]`, are output from the `Edge Compressor`. The whole of each (68×68) `layer[0]` image represents the foveal patch from the original (544×544) image captured by the camera. On `layer[1]`, this foveal patch is only the inner 34×34 square. On `layer[2]` and `layer[3]`, respectively, the foveal patches reduce to the inner 17×17 and 9×9 squares.

Because of concern that the amount of information available in these fovea might be too low to yield accurate tracking information, the KITTEN prototype arbitrarily defines an extended fovea to include the whole (68×68) `layer[1]` image. Thus, on `layer[2]` and `layer[3]`, the actual foveal patches searched are the inner 34×34 and 17×17 squares.

Movement detection operates independently (and in parallel) on each of the four layers. For each frame at time t, two copies shifted respectively left and

right by one pixel are made. These three images (the original plus the two copies) are compared against three other images: the frame at time (*t* - *1*) plus two copies shifted (respectively) up and down by one pixel.

In the *occam* KITTEN implementation, the horizontal shifts are achieved by logical one-bit shifts. The duplicated boundary regions in the compressed bit-mapping for each line mean that no bits need to be shifted between words. Vertical shifts are implemented by *occam* abbreviations. The comparisons are (one-cycle) *exclusive-or* operations followed by full-word `BITCOUNT` instructions. The extra-foveal regions are excluded by logical *and*-masks (horizontally) and by suitable index boundaries (vertically). The *and*-masks also exclude unwanted (horizontal) duplicates. All this is easy and efficient to express in *occam*, with full anti-alias checking and no assembler inserts.

A 'significant' minimum from the 3×3 comparisons between successive frames indicates the movement of a 'significant' object. A *one*-pixel movement (up, down, left or right) detected on layer i corresponds to a 2^i-pixel movement in the original camera image. Thus, the different layers of image processing respond to different scales of movement.

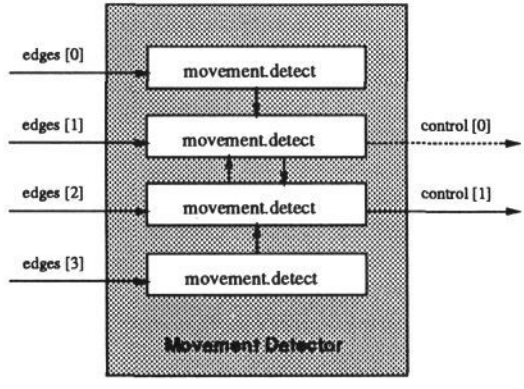


Figure 3: Movement Detection Architecture

The movement detectors for each layer are handled by separate transputers — see Figure 3. Because we only restrict our attention to the foveal region in each image, `layer[0]` and `layer[1]` have the same work-load, but the duties become increasingly lighter lower down the stack. To make up for this, `layer[2]` and `layer[3]` are programmed to conduct wider searches for movement. In fact, `layer[3]` has time to shift each foveal image by (plus and minus) three pixels in each axis and compute 7×7 comparisons.

It is important not to accept just any minimum from the computed comparison matrix at each layer. A minimum sufficiently close to the average may just be the result of 'noise' in the images and not indicative of any real object movement. The test for ' significance' adopted is only to accept a minimum that is more than k standard deviations below the average for the matrix, where k is a 'tuning' parameter set by experiment. During tracking, this threshold needs to be highest for `layer[3]` (which has the least amount of information from which to compute movements). Again, Figure 3 has been simplified by omitting the reverse-flowing control channels that are used to adjust k (or pass back other control signals destined for earlier stages in the pipeline).

The movements detected by `layer[0]` (±1 pixel) are passed down to `layer[1]`. The movements from `layer[3]` (±8, ±16 and ±24 pixels) are passed up to `layer[2]`. `Layer[1]` (detecting ±2 pixels) and `layer[2]` (detecting ±4 pixels) then exchange all their information, select the largest absolute movements horizontally and vertically and output these. These outputs are, of course, identical. One of these is connected to the next stage in the pipeline (`Tracking Control` — see Figure 1). The other (dashed in Figure 3 and not shown in Figure 1) is connected in our prototype system to a display manager process (so we can see what the system is doing!).

6 Tracking Control

Tracking is implemented using just one *transputer* that manages the hardware interface to the camera control system. DACs and ADCs (for feedback information) are memory mapped on to *occam* `ports` that enable absolute positioning of pan and tilt settings. Camera zoom, focus and gain may also be adjusted.

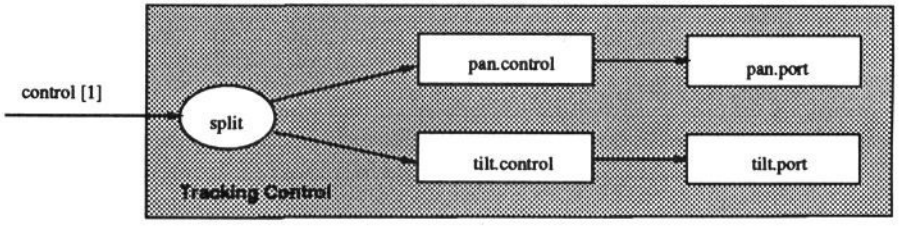


Figure 4: Tracking Architecture

Figure 4 shows the software architecture that controls the pan-and-tilt motors. The `split` process de-multiplexes the incoming movement (or initial position) information on to separate pan-and-tilt channels. The `pan.control` and `tilt.control` processes are identical (modulo some 'gearing' tables), as are the `pan.port` and `tilt.port` processes (modulo the actual DAC/ADC ports).

The gearing ratio depends on the level of zoom applied to the camera lens — the greater the zoom, the smaller must be the camera movement that corresponds to a detected object movement (measured in pixel displacements). The movement spans of our pan and tilt motors are not identical — i.e. the pan and tilt gearings need to be different.

An estimate is maintained of the current (angular) pan-and-tilt velocities of the tracked object. Change reports, arriving 25 times per second from the `Movement Detector`, are treated as accelerations and integrated to yield current velocities. These are, in turn, integrated once more to produce absolute pan-and-tilt coordinates that are output to the DACs. This final integration cycles at 100 outputs per second because this gives smoother pan-and-tilt motion, especially for slower velocities.

An object whose velocity has been correctly estimated and which does not accelerate or brake will be held in the centre of the field of view (the fovea). So long as this object occupies the majority of the fovea, the movement detectors will report zero accelerations (i.e. the moving background will be ignored) and tracking will be maintained. If the estimate were too high or the object brakes,

negative accelerations will arrive to correct the tracking. If the estimate were too low or the object accelerates, the tracking will again be suitably modified. Pan and tilt movements are tracked independently and in parallel.

The accelerations reported by the movement detectors are 'noisy' approximations to the true changes in velocity of the object. The control processes only receive discrete values from the set $\{0, \pm1, \pm2, \pm4, \pm8, \pm16, \pm24\}$ and a percentage of these will be completely wrong! When an incorrect value is processed, the camera starts to move away from the tracked object. The image of this object starts to move in the fovea and this generates correcting signals that are fed back to `Tracking Control`. Depending on the latency within the loop and the percentage of erroneous movement detections, this feed-back control can maintain tracking despite the apparent paucity of information. In KITTEN, the latency and error rate are sufficiently low for the tracking to work well — see [7].

7 Discussion — a Lesson From Biology

KITTEN demonstrates a practical real-time image processing application whose computational base is provided entirely by a small *transputer* system. No floating-point support is needed either from DSP or vector-processing engines — or even from T800s! The bulk of the computation involves only low-level, but massively replicated, logical operations: *addition*, *greater-than*, *and*, *exclusive-or*, *shift* and *bit-count*. Higher-level integer arithmetic, like *multiplication*, is relatively infrequent.

The design follows physiological models evident within the mammalian retina and visual cortex. The data from each captured image is (intelligently) reduced to about 1% of its original volume, before being analysed for movement. The reduction method exploits the fact that the images form a moving sequence and does not attempt to operate on individual frames. The detected movements are fairly 'noisy', but the overall feedback-loop ensures that errors are self-correcting and allows (reasonable) tracking to be maintained.

It is interesting to note how certain characteristics (such as rapid eye *dithering*), that appear at first glance to be physiological 'errors' that we should not bother to emulate, turn out to play a useful role. *Dithering* allows a cheap way for extracting edge information from stationary objects (e.g. it enables us to 'see' the object being tracked). It also provides a more complete response from naturally-shaped moving objects (by showing up edges even if they are parallel to the direction of movement). When tracking a target that has stopped moving, *dithering* helps keep the camera 'locked on' (because having the edge-information on all objects continuously available means that the correction response to dampen out erroneously reported movements — or deliberate 'feints' — can be much faster).

This paper describes only the *tracking* algorithm used within KITTEN. All the control processes actually operate in two modes — *passive* and *tracking*. In *passive* mode, the camera is held still and movement is sought from all parts of the images — including peripheral fields of view. Once a sufficiently large movement is located, the camera is moved to 'acquire' the target on its fovea and the system switches to *tracking* mode. Further details on these mode switches may be found in [8].

The 'intelligence' within KITTEN is at a fairly low level. The system is, after all, only emulating 'early' vision processing within the visual cortex. Tracking may well benefit from further correction signals fed back from 'higher-level thoughts' (e.g. for locating the tracked object more centrally in the fovea, choosing which of several moving objects to track, deciding when to lose interest in a tracked object, ...).

Clearly, the edge-filtered images could be delivered to other processing streams (e.g. for object recognition), that run on other *transputers* in parallel with this tracker pipline. In fact, the KITTEN prototype already delivers these images to a graphics processor with no degradation to its tracking capability. The message here is that extra functionality can be grafted on to the system by applying extra processors, without weakening the performance of its original task. Indeed, a response such as tracking may well benefit from multiple sources of feedback control — even if these signals are sometimes contradictory. Of course, the highest-level image processing system available to us is human. KITTEN could be made to interact well with such an operator.

At present, the implementation of the KITTEN prototype is largely unoptimised. We are using several — possibly 4 — more *transputers* than will finally be needed just to manage the tracking response! The prototype is heavily instrumented with information gathering (and filtering) processes that allow us to observe in real-time what is really happening inside the system. We also have on-line manual control of various system parameters (e.g. edge-detection thresholds, minimum-significance thresholds, gearing ratios, ...) to allow us to determine their best settings. Because we did not want such instrumentation to affect unduly the real-time performance of the system, some spare processing capacity was reserved in each processor. This instrumentation will not be needed in production versions of KITTEN.

The availability of T9000 *transputers* will have a considerable impact on the performance and economics of KITTEN. Its current capabilities could be performed by just two processors — one of which would require no external memory! Because of the physically shortened pipeline, latency would automatically be (at least) halved. Alternatively, we could revert to six processors, possibly five of which would need no external memory, and go for higher quality tracking (by processing more information to achieve lower error rates and detect higher accelerations of the target).

The aim is to produce a KITTEN system with tracking abilities that match those of a real kitten! At present, KITTEN will follow a person walking at a distance of about ten feet from the camera — or a coffee cup at a distance of three feet (moving with similar angular velocities/accelerations to the walker) [7]. In both cases, the camera zoom level is set to render the size of the tracked object to be about one fifth of the camera image.

Applications for such tracking systems are widespread. Potential large markets include (semi-)intelligent TV cameras, automobile guidance systems, surveillance cameras, docking control for in-flight refuelling, general target interception, video telephones and novel man-machine interfaces.

An important feature of all these image-tracking applications is that, given an external light source, they are all *passive* systems. Neither the tracker nor the object being tracked needs to be equipped with active radiation emitters (e.g. light, heat, radio, radar, sound, ...). This considerably enhances their practicality and security and lowers their costs.

8 Acknowledgements

The work reported in this paper complements research into Active Vision Systems by the University of Kent at Canterbury (Computing Laboratory) and the Defence Research Agency (RARDE, Fort Halstead, Kent). The basic notions of layered image sampling and motion detection may be found in [2, 3, 4]. with direct implementation details in [5, 6]. The 'short-cutting' ideas of dithered sampling, difference edge-filtering, bit-compression and velocity (rather than position) tracking are outside the standard stack model of vision. Further details may be found in [8].

We are especially grateful to Nigel Haig, Ian Moorhead and Richard Clement (from RARDE) and to Andrew Smith, Vedat Demiralp, Colin Willcock, Gordon Makinson and John Southall (from UKC) for countless lengthy debates about all these matters. A recent devious idea to use the *transputers*'s '2-D block move' instruction to speed up dithered image sampling (and, thereby, reduce feedback latency) is due to Tony Debling of INMOS — to whom thanks!

The prototype KITTEN system has been developed and implemented on the MEiKO Computing Surface at the University of Kent.

References

[1] R.H.S.Carpenter, **Movements of the Eyes**; published by Pion Ltd., London; ISBN 0-85086-109-8; 1988.

[2] P.J.Burt, **The Laplacian Pyramid as a Compact Image Code**; IEEE Transactions on Communications, pp. 532-540; 1983.

[3] G.J.Burton, N.D.Haig and I.R.Moorhead, **A Self-Similar Stack Model for Human and Machine Vision**; in Biological Cybernetics (53), pp. 397-403; 1986.

[4] I.R.Moorhead and N.D.Haig, **A Stack Model of Vision with some Pre-attentive Properties**; in 'Conference Proceedings of MARI 1987'; AGPB, Paris; 1987.

[5] A.B.Smith and P.H.Welch, **Real-Time Transputer Models of Low-Level Primate Vision**; in Proc. of the 11th OUG Tech. Conf., Univ. of Edinburgh, Scotland; edited by J.Wexler; pp.171-181; IOS Press, the Netherlands; ISBN 90 5199 11 1; Sept., 1989.

[6] A.B.Smith and P.H.Welch, **A Transputer Based Active Vision System**; in Proc. of the 15th WoTUG Technical Conference, University of Aberdeen, Scotland; edited by A.Allen; pp.112-121; IOS Press; ISBN 90 5199 085 5; April, 1992.

[7] P.H.Welch, **Silicon Retina II: a Foveal Tracking System (KITTEN)**; video (running time 25 minutes); the University of Kent; October, 1991.

[8] P.H.Welch and D.C.Wood, **KITTEN — A Foveal Image Tracker**; in 'Image Processing and Transputers'; edited by H.Webber, IOS Press; July, 1992.