Attentive Visual Tracking

J. Roberts and D. Charnley
Advanced Systems Research Group
Dept. of Aeronautics and Astronautics
University of Southampton
Highfield, Southampton, SO9 5NH, U.K.

Abstract

The research reported here addresses the problem of detecting and tracking independently moving objects from a moving observer in real time, using corners as object tokens. Local image-plane constraints are employed to solve the correspondence problem removing the need for a 3D motion model. The approach relaxes the restrictive static-world assumption conventionally made, and is therefore capable of tracking independently moving and deformable objects. The technique is novel in that feature detection and tracking is restricted to areas likely to contain meaningful image structure. Feature instantiation regions are defined from a combination of odometry information and a limited knowledge of the operating scenario. The algorithms developed have been tested on real image sequences taken from typical driving scenarios. Preliminary experiments on a parallel (Transputer) architecture indicate that real-time operation is achievable.

1 Introduction

Feature detection and tracking are fundamental to visual sensing in tasks such as establishing scene content and image data structure for vehicle navigation, localisation and map building. Most algorithms for visual tracking assume that the sensor (camera) is mounted on a moving platform operating in a static environment, to instantiate and maintain feature trajectories. This restrictive assumption may be acceptable for well-defined situations such as indoor environments, but is less useful for unstructured outdoor scenarios which may well contain multiple independently moving objects (e.g. other vehicles). The research reported here develops a strategy for the tracking of features which relaxes this constraint, allowing for situations where *both* the sensor and the environment are moving.

In vehicular applications, the source image signal contains far more data than can be analysed by a practical vision system in real time, and information of importance tends to be grouped in space as objects of interest. Most data available in the image signal is irrelevant to a vision task and it is important,therefore, to focus analysis on the critical regions and events, and ignore the majority of the signal that is not relevant[8].

The distinction between information containing regions and the rest of the visual field is solely in how the information from those regions is processed, if it is to be processed at all[1]. The potential reduction in the amount of data to be

[1]This selection of region-of-interest(s) is referred to as *covert attention*[8]

processed can lead to significant gains in performance. The tracking algorithm developed here employs just such a strategy, by using knowledge of visual sensor motion to constrain the search region for new features entering the field-of-view and local image-plane constraints for existing tracked features. Shapiro *et al.*[6] use a similar strategy, but perform feature detection over the entire image plane. The algorithm described here is novel in that *global* feature detection is abandoned and only performed in the relevant regions-of-interest. The system is therefore *data-driven* at a very low-level.

Conventional full image correspondence and structure-from-motion algorithms were not developed with parallelism specifically in mind[4, 10]. Consequently their performance is limited when implemented in parallel[9]. New algorithms (reliant only on 2-dimensional information), more amenable to effective parallelisation, are presented here.

2 The tracker

2.1 Region-of-interest tracking

In order to visually track objects in the real world, meaningful features - those generated by 3-dimensional object/scene structures - must be extracted from an image sequence. Corners, unlike edges, are discrete in 2 dimensions in the image plane allowing unambiguous motion estimates; they are often more abundant than straight edges in the natural world making them ideal features to track in an outdoor environment. We have therefore chosen to track corners, with detection achieved using an existing corner detection algorithm [3].

A new approach is adopted here in using data-driven, *local* (rather than global) feature detection[6, 7], where corner detection is performed over the entire image plane; we only perform the corner detection in the areas of the image-plane where we *expect* to find corners. This has been shown to reduce computational overheads by a factor of 5 typically when processing outdoor image sequences (approximately 500 corners) as only a 1/5th of the image-plane is being processed.

2.1.1 Tracking philosophy

One of the vital problems in motion analysis is how to match a set of features over an entire image sequence. This problem is known as the *correspondence problem*, and is solved *here* by assigning an individual "tracker" to each corner. Trackers subsequently search for their corners in a small region-of-interest around the corner's next predicted position. This strategy is similar to the one used by Shapiro[6] except that we only perform corner detection within the searched region-of-interest. This use of individual, independent trackers means feature tracking may readily be performed in parallel.

Assume that a corner has been detected at time t, and is located at a position $\mathbf{r}(t)$, where $\mathbf{r}(t) = (x(t), y(t))$. With only a single observation we have no notion of the corner's image-plane velocity ($\dot{\mathbf{r}}(t)$) and hence have no idea where to search for it in the next frame. If we therefore assume that features do not move "too far" between successive frames (a reasonable assumption for video rate processing) we may place a *region-of-interest (ROI)* in frame $t+1$ around its image-plane position found in frame t. Corner detection is then performed within this region to produce a list of candidate corners. A selection process (matcher, see Section

2.1.2) is therefore required to resolve any conflicts and match the *correct* candidate corner to the tracker's corner (Figure 1(a)).

Assuming that a successful match is made between frames t and $t+1$, it is possible to *predict* the corner's image-plane position in frame $t+2$. A simple constant image-plane velocity predictor, rather than the model-based Kalman filter often used, proved sufficient for the video-rate sequences considered (Equation 1). The size of the search region may now be reduced since we have more confidence in the corner's likely position (see Figure 1(b)).

$$\mathbf{r}(t+1) = 2\mathbf{r}(t) - \mathbf{r}(t-1) \tag{1}$$

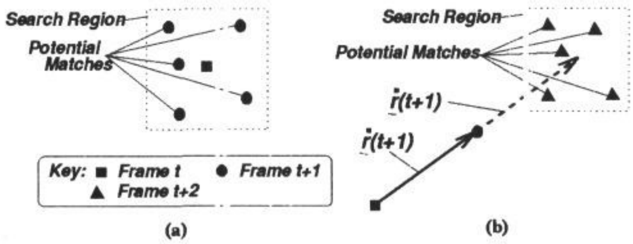

Figure 1: Position of (a) initial search region and (b) subsequent search region.

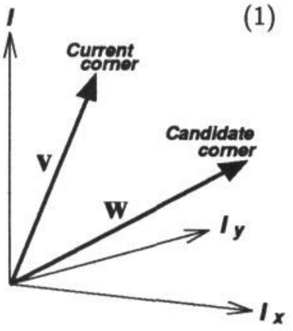

Figure 2: Match vectors

2.1.2 The matcher

The matcher uses the image-plane attributes of each feature-point, generated at the time of feature extraction, in an attempt to avoid the relatively expensive template matching techniques[6]. The pixel intensity and image x and y-gradients are used as components of the attribute vector (Figure 2). A match confidence value $m(\mathbf{v}, \mathbf{w})$ may then be calculated by comparing the normalised magnitude of the difference vectors between each candidate corner vector ($\mathbf{w}$) and the tracker's current corner vector ($\mathbf{v}$):

$$m(\mathbf{v}, \mathbf{w}) = \frac{\mid \mathbf{v} - \mathbf{w} \mid}{\sqrt{\mid \mathbf{v} \mid\mid \mathbf{w} \mid}} \tag{2}$$

The candidate corner which has the minimum value of $m(\mathbf{v}, \mathbf{w})$, as long as it is below a predefined threshold, is then declared the best candidate with its attributes and image-plane position being transferred to the tracker.

2.2 Corner instantiation

While significantly increasing the efficiency of the feature extraction process, the region-of-interest strategy introduces the problem of how to find *new* scene structure entering the field-of-view. This problem of finding new structure does not occur in "global" algorithms (e.g. the DROID system[7]) relying on global feature detection because those features remaining unmatched are instantiated as new structure. In contrast we search for new features only in areas of the image-plane likely to give rise to new information. Dynamic feature instantiation regions are defined from a combination of odometry information and a limited knowledge of the operating scenario.

2.2.1 Features from trackers

Areas of the image-plane likely to contain new or previously undetected scene structure are those areas near features currently being tracked. Hence, any unmatched candidate corners remaining in the tracker's region-of-interest (Section 2.1) that are also *stronger* corners than the one selected for the match are assigned to newly instantiated trackers for tracking in subsequent frames. This allows new corners to be generated on objects of interest as they resolve in the image. It is also possible to *associate* features, and hence *split* tracks simply by retaining information regarding the source tracker of a newly instantiated tracker[1]. The reverse problem of structures *merging* is discussed Section 3.1.

2.2.2 Focus-of-expansion

Consider for example a driving scenario, in which an observer translates forwards. Stationary (or distant) objects in the environment appear to move along paths radiating from the *focus-of-expansion* (FOE) New image structure is likely to be observed around the FOE making it a good region in which to instantiate new corners.

In this scenario, the real world positions of image data located near the FOE in the image plane are typically hundreds of metres away from the camera. This results in any features, such as corners, being difficult to resolve, due to the *low resolution*. This low resolution is simply caused by the uniform quantisation of the image, into pixels, which results in the loss of detail of distant objects. The result of this low resolution is that the area of the image-plane immediately around the FOE does not contain much useful, or reliable corner information.

In the driving scenario corners must be instantiated on objects (independently moving objects, or the static world) that are no nearer the observer vehicle than a certain *safe* distance. This safe distance, or *minimum detection distance*, D_1, would be based on the stopping distance of the observer vehicle, which in turn is a function of the vehicle's velocity, $V_{observer}$. If we assume that the observer is moving on a planar road surface, and is traveling down a tunnel of rectangular cross-section[2], with the FOE at the tunnels vanishing-point, then we may define an *outer FOE window* at a distance D_1 in the real world. As the observer vehicle moves into the scene, down the imaginary tunnel (see Figure 3), all new image data contained within the imaginary tunnel in the static world will flow out of this outer FOE window.

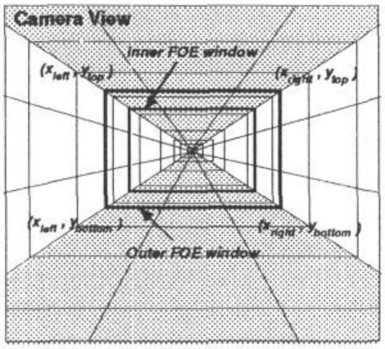

Figure 3: Position of FOE windows in the image-plane.

Since we have defined a minimum distance by which time we wish to have instantiated new corners from the static world, we do not need to search for new corners that are located further away from the observer than D_1. In other words, if we could sample at a high enough rate, and ensure that corners only moved a maximum of a pixel between successive frames, corner instantiation could be restricted to the pixels that make up the outer FOE window frame (i.e. a frame 1 pixel thick). However, this is not the case. If we know at what velocity the observer is moving and we specify how many times we wish to have seen a feature by the time it reaches a distance D_1 (say n times), we may calculate how far a feature in the static world will move in n frames in the image plane. It is then possible to define an *inner FOE window* (see Figures 3 and 4(b)), at a distance D_2 in the real world.

$$D_2 = D_1 + V_{observer} n \Delta t \qquad (\Delta t \text{ is the time between frames}) \tag{3}$$

Feature instantiation is then performed in the region between these two FOE windows. As mentioned above, the immediate area around the FOE itself does not contain reliable corner information due the effective low resolution of distant objects. The fact that we do not need to search for corners to instantiate within the inner FOE window overcomes the low resolution problem, assuming that the inner FOE window totally encloses the low resolution region.

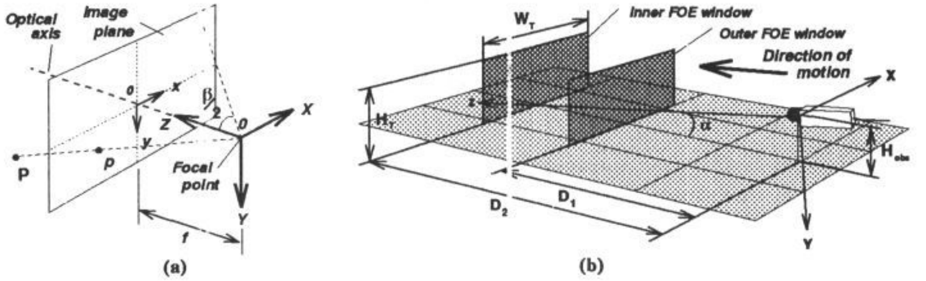

Figure 4: (a) Axis system (b) position of FOE windows in real world.

Referring to Figure 4(a), the 3D point $\mathbf{P} = (X_p, Y_p, Z_p)$ in the real world projects to $\mathbf{p}$ in the image-plane, where $\mathbf{p}$ is given by

$$\mathbf{p} = \left(\frac{fX_p}{Z_p}, \frac{fY_p}{Z_p}\right) = (x_p, y_p) \qquad (f = \text{sensor focal length}) \tag{4}$$

If α is the camera look-down angle, H_{obs} the height of the camera above the ground plane (assumed flat), and W_T and H_T the width and height of the driveable tunnel respectively, then the position of the FOE windows in the image-plane is given by Equations 5 and 6.

$$x_{left} = -f\frac{W_T}{2D}, \qquad x_{right} = +f\frac{W_T}{2D} \tag{5}$$

$$y_{top} = f\frac{(H_{obs} - H_T)}{D}, \qquad y_{bottom} = f\frac{H_{obs}}{D} \tag{6}$$

where $D = D_1$ or D_2 depending on the FOE window, and α is small.

2.2.3 *Image borders*

The FOE ROI takes care of new image data entering the field-of-view as a result of forward translations of the camera. However there are many situations where image data will enter the field-of-view from regions other than the FOE. In the driving scenario an obvious source of new image data is at the vertical edges of the image-plane, where we would first expect to see any over-taking vehicles for example. Placing instantiation regions at the left and right-hand sides of the image-plane therefore ensures acquisition of any features due to over-taking vehicles.

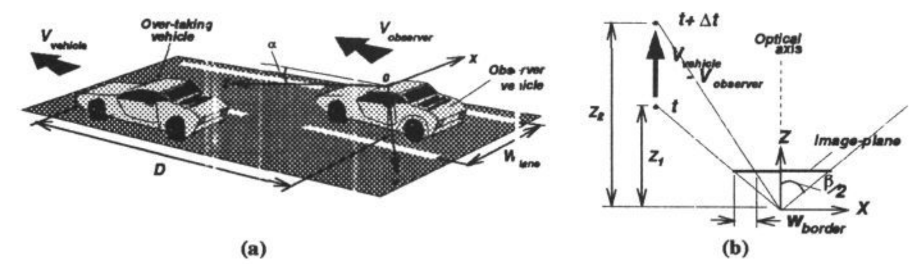

Figure 5: (a) An over-taking vehicle, (b) geometry and axis system.

The width of the border ROI is determined by the maximum distance we expect a new feature entering the field-of-view to move in a single frame time. If d_1 is the distance of the vehicle feature from the observer at time t, then the distance of the vehicle feature from the observer at time $t + \Delta t$, d_2 is given by

$$d_2 = d_1 + \Delta t(V_{vehicle} - V_{observer}) \tag{7}$$

where $V_{vehicle}$ is the maximum expected velocity of an over-taking vehicle in the z-direction. If we assume that α is small, then the width of the border ROI, w_{border} is given by

$$w_{border} = fW_{lane}\left(\frac{1}{d_1} - \frac{1}{d_2}\right) \tag{8}$$

If β is the camera field-of-view in the x-axis (Figure 4(a)), then an over-taking vehicle feature just entering the field-of-view will be a distance d_1 away from the observer, where d_1 is given by

$$d_1 = \frac{W_{lane}}{tan\left(\frac{\beta}{2}\right)} \qquad \text{(assuming } \alpha \text{ is small)} \tag{9}$$

It must be noted that analogous results for FOE/border regions can of course be developed for large look down angles, α. The equations given here are for illustration only.

In practice, the border regions are moved away from the image-plane edges slightly. This is done because image data entering the field-of-view with high relative velocities tend to be blurred (due to low camera shutter speeds). The simplest way of recalculating the border's position is to use the camera's portion of the field-of-view that is non-blurred, β', and use the equations above. Note that β' is solely a function of the camera's shutter speed and the expected maximum relative velocity observed.

It must also be noted that any observer yawing motion will also cause new static scene structure to enter the image-plane from the sides. Hence, dynamic yaw border regions may be developed in a similar fashion. The positions of the FOE regions will also change due to observer yaw motion.

2.3 Instantiation and Tracking in real image sequences

The feature generation and tracking algorithms were implemented and tested in sequential form on a Sun workstation. Figure 6 shows the corner trajectories produced by a focus-of-expansion ROI after 85 frames of the *village* sequence (approximately 4 seconds). The camera was mounted on a vehicle, and was driven along a road at approximately $20kmh^{-1}$.

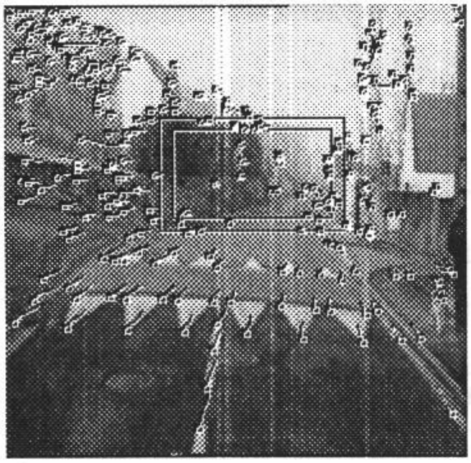
Figure 6: Frame 85 of the *village* sequence, FOE ROI only.

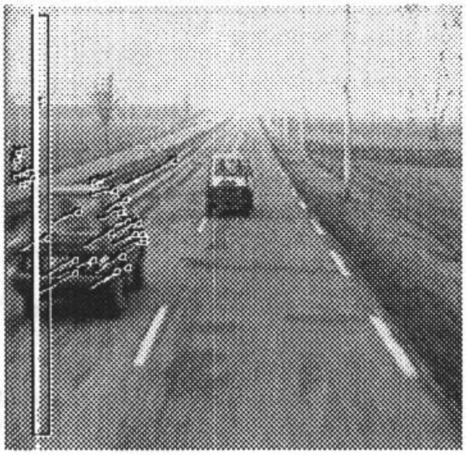
Figure 7: Frame 27 of the *motorway* sequence, Border ROI only.

Image data[1] were initially smoothed using a Gaussian filter ($\sigma = 0.5$). Dark lines represent the past positions (last 5 frames) in the image plane of the corners - the *trajectories*; small white boxes show each corner's current position. The FOE ROI is also indicated. Only corners that have been tracked consistently for the previous 5 frames or more are shown; corners that have been tracked less consistently were considered, as yet, unreliable and are not displayed. It can be seen from the figure that the FOE ROI alone produces a good spread of trajectory information across the image-plane, as would be expected when translating forwards into the scene. It may be noticed that a number of corners are found within the inner FOE region. These are produced as a consequence of tracker ROIs instantiating new features, some of which may occasionally occur inside the inner FOE region.

A border ROI has been used on a *motorway* sequence in order to pick-up any vehicles entering the field-of-view. Figure 7 shows frame 27 of the sequence, and for clarity, only corner trajectories produced by the border ROI are shown. The border ROI has clearly detected the over-taking vehicle.

These results illustrate that constrained data-driven feature instantiation developed here can adequately produce sufficient feature trajectory information for extraction of both the static background and independently moving objects.

3 Parallel implementation

Real time visual tracking demands high computational effort. Feature generation and tracking, as considered here, has been designed with parallelisation in mind. A parallel model is described which exploits the natural parallelism inherent in these particular algorithms.

In the proposed tracking algorithm, each corner is tracked independently, making processor farming a natural choice of architecture since it is suitable for situations in which the processing problem may be broken up into multiple independent tasks. When work starvation can be avoided, processor farming is inherently load balanced.

[1] Image sequences used are from the PANORAMA ESPRIT project.

3.1 System architecture

There are four different processor types in the architecture; the Corner Store, Controller, Image Store, and Tracking Engin›s (see Figure 8).

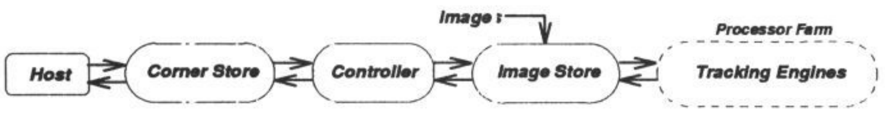

Figure 8: Overall sy:tem model.

• **Corner store.** The corner store's main function is to store the current and past positions of the corner features (i.e. it maintains feature trajectories). The corner store remains idle until a request is received from the controller processor for a corner to be tracked. The attributes, and predicted image-plane position of the corner are then immediately sent to the controller processor.

It is also the corner store's task to detect the presence of duplicated trackers. Because trackers are completely independent of one another and of instantiation regions, it is possible that some features will be tracked by multiple trackers simultaneously. As results (the updated position of the features) come in via the controller, a note is made in the *blackboard* cf the feature's ID and its position in the image-plane. When there is more than one feature at a single image-plane location, a decision is made and the appropriat› trackers are terminated leaving the single most reliable tracker to continue. This decision is based on the similarity of the features' trajectories over the subsequent two frames. In this architecture, the corner store is located on the root processor making user access to the corner store relatively simple. It is also intended that oth›r processors performing higher-level tasks will be attached directly to the corner :tore.

• **Controller.** The controller is the hear; of the system controlling the flow of data from the corner store to the tracking engines, via the image stores. Control is achieved with the aid of two buffers, the *free ·vorker buffer*, which contains a list of tracking engines currently awaiting work, and the *work buffer*, which contains the work for the tracking engines, corner data t› be tracked (i.e. corner attributes, image-plane position, etc). The controller also acts as a result router between processor farm and the corner store. Any results received by the controller from the processor farm are immediately forwarded to the corner store, with the source of the result, the ID of the tracking engine from which the result originated, being noted as an entry into the free worker buffer (a result implies that that tracking engine must be free). New work may then be sent from the work buffer, via the image stores, to a free tracking engine.

• **Image store.** The image store contains the actual image and can be thought of as a simple filter. As corner data passes through the image store, the required image patch is extracted from the image, re-packaged with corner data and sent on to the tracking engine processor farm. Like the controller, the image store routes results back from the tracking engines. It may be noted that the image store may consist of a number of individual processors, with the image data distributed among them. This allows simultaneous access to different areas of the image-plane.

• **Tracking engines.** The tracking engines perform the actual corner detection and matching. In addition, they must also route corner data destined for other tracking engines in the processor farm and pass results back to the corner store via the image store and controller.

3.2 Performance

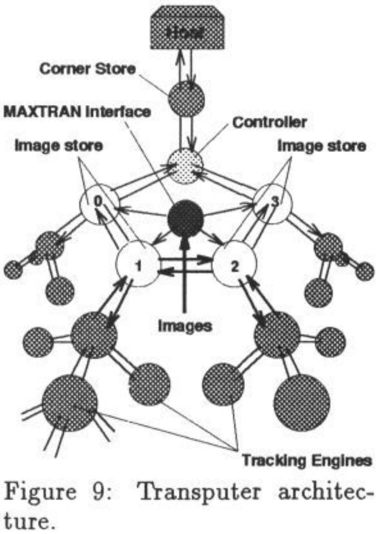

Figure 9: Transputer architecture.

The architecture described in the previous section was implemented and tested on 30 T800 Transputers (20MHz processors, 10MHz links). Four image store processors were used leaving 24 tracking engines. The four image stores were configured as a ring with individual tracking engine processor farms attached to each image store processor (Figure 9). Performance testing involved tracking a number of corners between two consecutive frames of an image sequence. The performance measure used was simply the time taken to track 64 corners. The distribution of image plane data among the image store processors means that the performance of the architecture is, to a limited extent, data-dependent; performance is a function of corner position in the image-plane. This is due to the way that data flows around the ring. Architecture performance may be improved (*optimized*) by targeting corners to the optimum image store based on the corner's x, y image plane position.

A comparison of the *non-optimized* and *optimized* performance is shown in Figure 10, where 64 corners were tracked. A performance of 11Hz was achieved tracking 64 corners using 24 workers. Note that *speedup* is defined as the time taken for n tracking engines/workers to track 64 corners divided by the time taken by a single worker (with a single image store) to track the same number of corners. It is clear from Figure 10 that near-linear speedups are achieved. The targeting of corners to image stores is also seen to have a positive effect when the number of tracking engines is greater than 20.

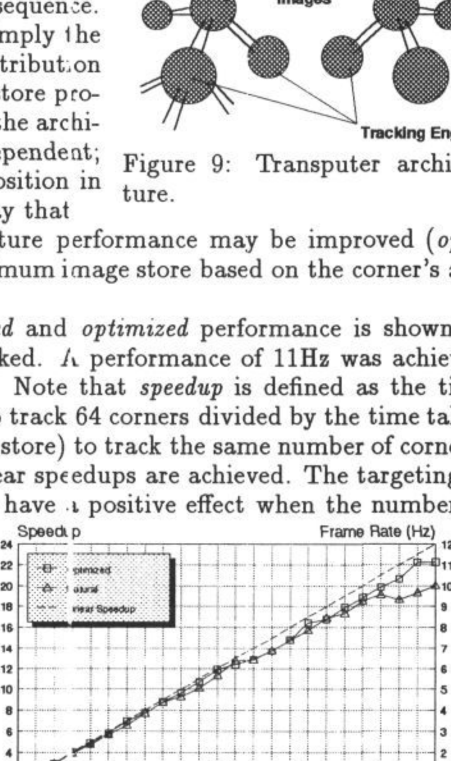

Figure 10: A comparison of non-optimized and optimized architecture performance.

The performance profile shows irregularities caused by the initial condition of the network (i.e. all workers free). This results in many workers finishing their work at about the same time, causing a number of workers to be free simultaneously. This effect is seen to subside with time. The architecture and its performance are discussed in greater detail in [5].

4 Conclusions

An algorithm has been developed for the detection of independently moving and/or deformable objects, in a static world from a moving observer. The algorithm presented is based on the concept of independent local trackers, using corner features as primitives. The technique is novel in that unlike traditional feature tracking algorithms, where feature instantiation is carried out over the entire image plane, here it is restricted to those areas, *regions-of-interest* (ROIs), most likely to contain new image data. It was found that two distinct types of instantiation ROIs were required for a typical driving scenario. The first type of ROI is the *focus-of-expansion* (FOE) ROI. This region instantiates new features brought into the image-plane as a result of forward translations of the camera, its size being determined by the observer's velocity. The second type of instantiation ROI is the *border* ROI. Border ROIs are required to instantiate new features entering the field-of-view laterally from for example, over-taking vehicles.

Independent feature tracking lends itself naturally to parallelisation. A parallel architecture was implemented to evaluate the performance potential which suggests that near-real time feature tracking is possible using this algorithm.

The work may be summarized as follows:

- A novel feature detection and tracking strategy has been developed where feature detection is only performed in areas of the image plane containing useful information.
- Static world and independently moving objects can be detected and tracked.
- Parallel implementation of the algorithm has shown that near-real time operation is possible and that full video rate (25Hz) performance is achievable.

The next task is to to use trajectory information for obstacle collision avoidance. This will involve the modeling and segmentation of the observed feature trajectories into distinct objects.

References

[1] Bar-Shalon, Y. and T.E. Fortmann (1988). Tracking and Data Association, *Mathematics in Science and Engineering, Vol. 179, Academic Press, INC*

[2] Enkelmann, W. (1991). Obstacle Detection by Evaluation of Optical Flow Fields from Image Sequences, *Image and Vision Computing*, Vol. 9, No. 3, pp. 160-168.

[3] Harris, C.G. and M.J. Stephens (1988). A Combined Corner and Edge Detector, *Proc. of the Fourth Alvey Vision Conference*, Manchester, pp. 147-151.

[4] Mirmehdi, M and T.J. Ellis (1993). Parallel Approach to Tracking Edge Segments in Dynamic Scenes, *Image and Vision Computing*,Vol. 11, No. 1, pp. 35-48.

[5] Roberts, J.M. and D. Charnley (1993) Parallel Visual Tracking *1st IFAC International Workshop on Intelligent Autonomous Vehicles, Southampton.*

[6] Shapiro, L.S., H. Wang and J.M. Brady (1992). A Matching and Tracking Strategy for Independently-moving, Non-rigid Objects, *3rd BMVC, Leeds*

[7] Stephens, M.J., R.J. Blissett, D. Charnley, E.P. Sparks, J.M. Pike (1989). Outdoor Vehicle Navigation using Passive 3-D Vision, *IEEE Conf. on Computer Vision and Pattern Recognition, San Diego, (1989).*

[8] Swain, M.J., M. Stricker *et al* (1991). Promising Directions in Active Vision, *NSF Active Vision Workshop, University of Chicago.*

[9] Wang, H. and J.M. Brady (1992). Parallel Implementation of DROID and Performance Evaluation, *Tech. Report, Robotics Research Group, Oxford University.*

[10] Zhang, Z. and O.D. Faugeras (1992). Three-Dimensional Motion Computation and Object Segmentation in a Long Sequence of Stereo Frames, *International Journal of Computer Vision*, Vol. 7, No . 3, pp. 211-242.