Learning the distribution of object trajectories for event recognition — Source link

Neil Johnson, David C. Hogg

Institutions: University of Leeds

Published on: 01 Jul 1995 - British Machine Vision Conference

Learning the Distribution of Object Trajectories for Event Recognition

Neil Johnson and David Hogg
School of Computer Studies
The University of Leeds
Leeds, LS2 9JT
United Kingdom
email: {neilj,dch}@scs.leeds.ac.uk

Abstract

The advent in recent years of robust, real-time, model-based tracking techniques for rigid and non-rigid moving objects has made automated surveillance and event recognition a possibility. We present a statistically based model of object trajectories which is learnt from image sequences. Trajectory data is supplied by a tracker using Active Shape Models, from which a model of the distribution of typical trajectories is learnt. Experimental results are included to show the generation of the model for trajectories within a pedestrian scene. We indicate how the resulting model can be used for the identification of incidents, event recognition and trajectory prediction.

1 Introduction

Existing vision systems for surveillance and event recognition rely on known scenes where objects tend to move in predefined ways (see eg. [1]). We wish to identify incidents, recognise events and predict object trajectories within unknown scenes where object behaviour is not predefined. We use an open pedestrian scene as an example of such a situation since pedestrians are free to walk wherever they wish.

In this paper, we develop a model of the probability density functions of possible instantaneous movements and trajectories within a scene. The model is automatically generated by tracking objects over long image sequences. The pdf's are represented by the distribution of prototype vectors which are placed by a neural network implementing vector quantisation. The temporal nature of trajectories is modelled using a type of neuron with short-term memory capabilities.

We indicate how the model can be used to recognise atypical movements and thus flag possible incidents of interest, and how attaching 'meaning' to areas of the distributions representing similar instantaneous movements and trajectories allows event recognition and trajectory prediction to be performed.

2 Data

It is assumed that raw data is available giving the 2D image trajectories of moving objects within the scene. For our experiments, we use an object tracker (Baumberg & Hogg [2]), based on Active Shape Models (Cootes *et al.* [3]) and acquired automatically from observing long image sequences (Baumberg & Hogg [4]). This system provides efficient real time tracking of multiple articulated non-rigid objects in motion and copes with moderate levels of occlusion. In our experiments, pedestrians are tracked in a real world scene using a fixed camera (eg. see Figure 1(a)).

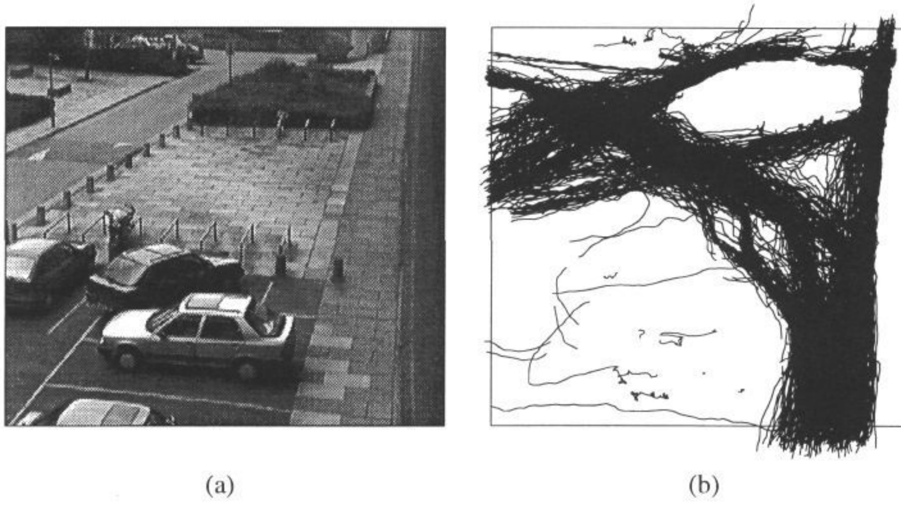

Figure 1: Raw data: (a) pedestrian scene, (b) raw trajectory data.

There is a one way flow of data from the tracker consisting of frame by frame updates to the position in the image plane of the centroid of uniquely labelled objects. The detection of atypical activity and the recognition of events is feasible within the image plane although it can also be carried out with trajectories that have been *back projected* onto the ground plane. The use of the image plane avoids introducing errors associated with the transformation of coordinates from the image to ground plane.

Since each new object being tracked is allocated a unique identifier, it is possible to maintain a history of the path taken by each object from frame to frame. The tracker processes frames at a fixed rate and thus, for an object i which has existed for n frames, we have a sequence T_i of n 2D image coordinates, uniformly spaced in time:

$$T_i = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\} \tag{1}$$

Figure 1 shows a large number of these raw data paths with centroid positions connected with lines (b) alongside an image of the 'empty' pedestrian scene from which they were obtained (a).

Instead of using a sequence of positions to describe an object's movements, we describe its trajectory in terms of a sequence of *flow vectors* where a flow vector f represents both the position of the object and its instantaneous velocity:

$$\mathbf{f} = (x, y, \delta x, \delta y) \tag{2}$$

Flow vectors are calculated from the raw data by considering the change in centroid coordinates between successive frames. Since the frame rate of the tracker is constant, these differences give us a measure of the instantaneous velocity of the object. Due to inaccuracies in the tracking process, the raw data will contain random noise. This noise is minimised by smoothing flow vectors over a moving window.

The velocity components are scaled relative to the positional components in order to balance their relative contribution when computing the similarity between flow vectors. The scaling factor is derived from the maximum observed object speed. Flow vectors are then transformed so that each component lies in the range [0, 1] (ie. $x, y, \delta x, \delta y \in [0,\ 1]$). Thus an object i which has existed for n frames is, after preprocessing, represented by a set Q_i of n flow vectors all of which lie within a unit hypercube in 4D *phase space*:

$$Q_i = \{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \ldots, \mathbf{f}_{n-2}, \mathbf{f}_{n-1}, \mathbf{f}_n\} \tag{3}$$

3 Modelling Probability Density Functions

In modelling the complex probability density function of N-dimensional vectors, we have two main aims:

- to form as concise and accurate a model as possible, and
- to enable ‘meaning’ to be attached to areas of the distribution.

One way of modelling the pdf would be to divide the feature space into an N-dimensional grid structure and increment a count for each cell whenever a vector falls within that cell. This would not be a concise model, and meaning would have to be attached to all cells. Instead, we model the pdf by the point distribution of prototype vectors using vector quantisation.

3.1 Vector Quantisation

Vector quantisation is a classical method of modelling pdf's by the point distribution of prototype vectors. We implement the technique using a competitive learning neural network which is taught in an unsupervised manner (see eg.[5, 6]).

Our network consists of a set of N input nodes (one for each component of the N-dimensional feature vectors) k output nodes (one for each prototype) and implements the following algorithm:

1. Randomly place the k prototypes within the feature space.
2. Initialise α, a monotonically decreasing gain coefficient in the range (0, 1).
3. Let $\mathbf{x}(t)$ be the input feature vector for this epoch.

4. Find the prototype $\mathbf{m}_c(t)$ which is nearest to this input by the Euclidean metric:

$$||\mathbf{x}(t) - \mathbf{m}_c(t)|| = min_i||\mathbf{x}(t) - \mathbf{m}_i(t)|| \tag{4}$$

5. Update prototypes as follows:

$$\begin{aligned} \mathbf{m}_c(t+1) &= \mathbf{m}_c(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_c(t)] \\ \mathbf{m}_i(t+1) &= \mathbf{m}_i(t) \;\; for\, i \neq c \end{aligned} \tag{5}$$

6. Decrease $\alpha(t)$ in line with a 'cooling schedule'.

7. Repeat steps 3-6 for many epochs.

After learning, each prototype will represent an approximately equal number of training feature vectors and the point density of the prototypes within the feature space will approximate the pdf of the feature vectors [6]. The model is thus more accurate in areas of high probability density and so the representation is both concise and accurate.

A modification to this algorithm to deal with sensitivity to the initial placement of prototypes is detailed in the Appendix.

In our network implementation, each output node represents one of the prototypes and is said to 'win' if it's prototype is the nearest to the feature vector being presented on the inputs. The output of a node i is calculated as follows:

$$O_i(t) = 1 - \frac{||\mathbf{x}(t) - \mathbf{m}_i(t)||}{\sqrt{N}} \tag{6}$$

Thus $O_i(t)$ decreases linearly from one to zero as the distance from $\mathbf{x}(t)$ to $\mathbf{m}_i(t)$ increases from zero to $\sqrt{N}$. The form of this output is not important until we add further layers to the network (described in Section 5).

The number of prototypes used to describe the distribution can be determined experimentally by calculating a *reconstruction error* [6] for different numbers of prototypes. A point is reached when increasing the number of prototypes does not significantly reduce the error.

4 Modelling the Pdf of Flow Vectors

A competitive learning network is used to model the pdf of flow vectors generated from the raw input data stream (see Section 2). Before the flow vectors can be presented to the network some further preprocessing is necessary.

As an object moves it sweeps out a continuous path in 4D phase space. This path is sampled at regular time instants to generate the sequence of vectors which is the result of preprocessing. When the speed at which the path is swept out is low, the sampled vectors are densely distributed, and when it is high, the vectors are sparsely distributed. This will result in a higher probability density in areas where the rate of movement along the path is low.

To avoid this problem, the path is resampled with a constant step size, δd. This generates a new sequence of flow vectors which are evenly distributed along the path. The value of δd is chosen to be as large as possible whilst still representing the detail of the trajectory.

A four input network can now be trained by sequentially presenting flow vectors generated in this way from a large number of object trajectories.

4.1 Experimental Results

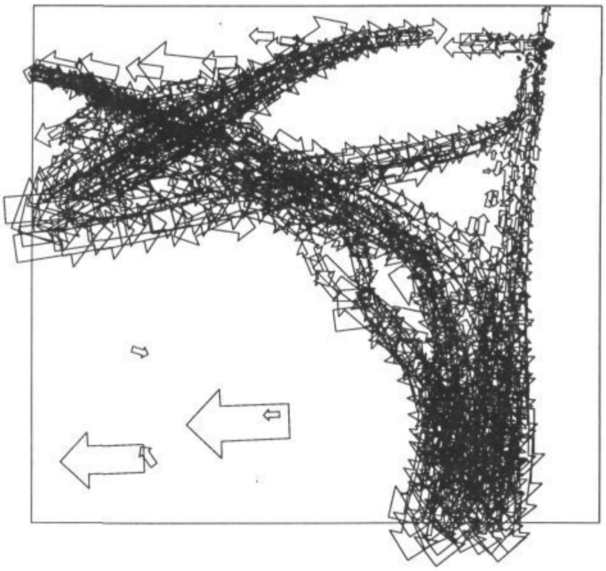

Figure 2: Distribution of prototypes in a 4 input, 1000 output node network trained on the trajectories shown in Figure 1(b).

The trajectories shown in Figure 1 (b) were used to train a network consisting of 4 input nodes and 1000 output nodes/prototypes. Flow vectors were generated using a factor of 20 in the scaling of velocity components over positional components. A value of $\delta d = 0.05$ was used for the generation of corrected flow vectors. The network was trained for 1000000 epochs with the gain coefficient α decreasing linearly from 0.999999 to 0.000001 over this period. A value of $\beta = 0.01$ was used for sensitivity adjustments (see Appendix).

The results of this experiment are shown in Figure 2. The prototype for each of the 1000 output nodes is displayed as an arrow, the position of which represents the (x, y) components, and the size and direction of which represents the $(\delta x, \delta y)$ components. Comparison between these prototypes and the raw trajectories shows the results to be plausible.

5 Modelling the Pdf of Trajectories

In order to model the pdf of sequences of flow vectors using a competitive learning network we need to form a representation of sequences with the following properties:

- sequences of different lengths are modelled.

- sequences which are similar should be close in the vector space of the representation and *vice versa*.

We model sequences of flow vectors by modelling the sequence of activations they cause on the outputs of the first network's competitive layer (Section 4). This reduces the set of possible sequences to those involving the flow vectors already discovered and is achieved by adding a further layer to the network developed in the last section. This layer consists of 'leaky neurons' and acts as a memory mechanism to record a history of activations.

5.1 Leaky Neurons

The leaky neurons used are similar to the Leaky Integrators of Reiss & Taylor [7] or the neurons of Wang & Arbib [8]. Leaky neurons are different to the neurons in most neural networks in that they hold a certain amount of their activation from previous epochs. This leaky characteristic is present in biological neurons where electrical potential on the neuron's surface decays according to a time constant. In this way the leaky neurons have a memory of previous activations.

A leaky neuron has a single input and a single output. The activation at epoch $t+1$ is calculated from the previous activation $a(t)$ and the current input I:

$$a(t+1) = \begin{cases} I & \text{if } I > \gamma a(t) \\ \gamma a(t) & \text{otherwise} \end{cases} \tag{7}$$

Where γ is a coefficient in the range (0, 1) which governs the rate of decay and thus the memory span of the neuron.

Such a neuron will mimic its input over a number of epochs unless the input decreases at a rate which is greater than the rate of decay of the neuron's activation. A leaky neuron with a slow decay rate (high value of γ) will thus retain a 'trace' of it's highest input.

5.2 Method

From equation 6, the output nodes of our competitive learning network produce an activation which decreases linearly from one to zero as the distance between the node's prototype and the input vector increases from zero to $\sqrt{N}$. As the flow vectors produced by a trajectory are presented to a trained network, the output of certain nodes (whose prototype the trajectory comes close to) will first increase and then decrease.

By connecting leaky neurons with slow decay rates to the output of these nodes, a trace of the trajectory will be formed in the activation of the leaky neurons. By connecting leaky neurons to the output of every node in a trained network we form a representation of the complete sequence of activation.

Sequences of any length can be represented up to a maximum defined by the number of prototypes and the memory span of the leaky neurons. Sequences must be simple (ie. the trajectory must not pass each prototype more than once) but this is almost always the case in reality. Since nodes whose prototypes are close in phase space will have similar outputs, the representation has a sense of the similarity between trajectories.

In order to approximate the pdf of trajectories we use vector quantisation to place prototypes within the vector space of the leaky neuron outputs, and thus model the pdf of these activations. Further work is required to assess the distortion to the pdf of trajectories caused by representing the trajectories in this manner.

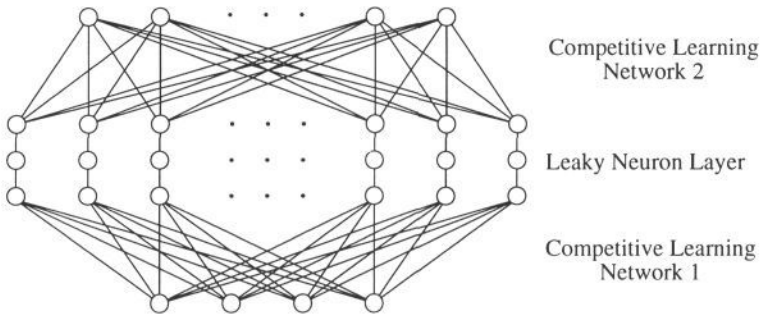
Competitive Learning Network 2
Leaky Neuron Layer
Competitive Learning Network 1

Figure 3: Architecture of multilayer network for approximating the pdf of flow vector sequences.

We implement this second vector quantisation by attaching a second competitive learning network to the leaky neuron layer (see Figure 3). In order to teach this second network, we sequentially present trajectories. For each trajectory we first zero the leaky neuron layer and then sequentially present the (uncorrected) flow vectors. When the whole sequence has been presented, the second network is taught on the activation on the leaky neuron layer. This process is repeated for many trajectories.

5.3 Experimental Results

A layer of 1000 leaky neurons was connected to the output nodes of the network trained in Section 4.1, the outputs of these neurons being connected to the inputs of a second competitive learning network consisting of 1000 input nodes and 100 output nodes. Flow vectors were generated as in Section 4.1 but sampling correction was not performed. A value of $\gamma = 0.99$ was used to govern the decay of activation in the leaky neurons. The second network was trained for 100000 epochs with the gain coefficient α decreasing linearly from 0.99999 to 0.00001 over this period. A value of $\beta = 0.1$ was used for sensitivity adjustments (see Appendix).

Some results from this experiment are shown in Figure 4. Figure 4(a) shows a representation of a prototype from the second network where the value of each component is displayed as a shaded arrow. The arrow indicates which prototype from the first network the component corresponds to and the shade represents the value (white being zero and black one). Figure 4(b) shows raw trajectories from the data set which cause the prototype represented in (a) to win. Figure 4(c) & (d) are as (a) & (b) but for another prototype.

Examination of prototypes and the raw trajectories for which they win suggests a plausible division of the feature space with prototypes representing trajectories covering different paths with differing velocities. Groups of trajectories with high probability density are represented by many similar prototypes as expected.

6 Event Recognition

The most obvious use for the model we have developed is assessment of the typicality of instantaneous movements and trajectories, where typicality is defined statistically. By

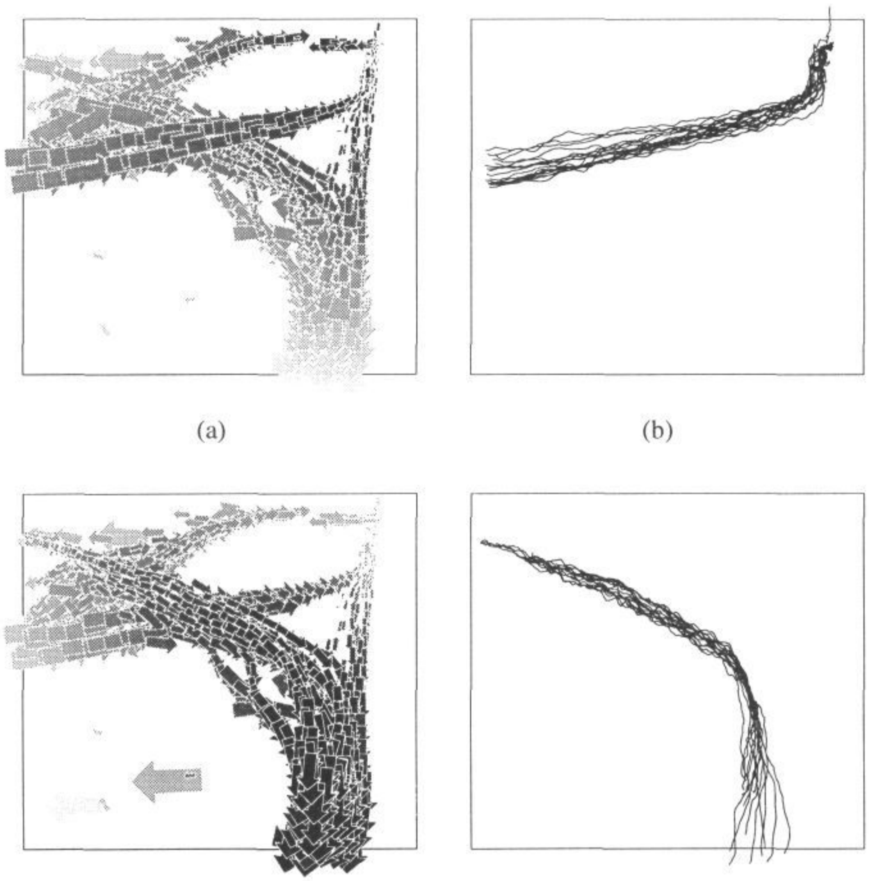

Figure 4: Trajectory learning results: (a) & (c) representations of two prototypes, and (b) & (d) raw trajectories which the prototypes represent.

observing the approximate probability density in the model of an object's instantaneous movements and trajectory, we can flag possible incidents of interest. In order to achieve this it is necessary to label each prototype with a value representing it's local probability density.

By estimating the volume v_i within feature space for which a particular node i wins, and assuming the probability density is constant within this region, the probability density can be approximated by

$$p_i = \frac{1}{kv_i} \tag{8}$$

Where k is the number of prototypes and the entire distribution is assumed to lie within a unit hypercube. Since estimation of v_i is impractical for high dimensional spaces, we can instead use the mean distance

$$D_i = \frac{\sum_{j=1}^{n} ||\mathbf{x}(t) - \mathbf{m}_i(t)||_j}{n} \qquad (9)$$

for which node i is the winner as a measure of relative probability density.

If partial trajectories are also learnt then continuous assessment of trajectory typicality is possible.

Recognition of simple and complex events can be achieved by attaching semantics or meaning to areas of the distributions. This is simply a matter of labelling the relevant nodes, and retrieving the information when the nodes are activated.

Trajectory prediction can be achieved in a similar way by labelling nodes who's prototypes represent complete trajectories with information acquired automatically in a further learning phase. Partial trajectories can then activate the node representing the most similar complete trajectory.

7 Conclusions

We have presented a statistically based model of object trajectories which is learnt from image sequences. The model is based on a neural network allowing fast parallel implementation. Experimental results show the generation of the model for the trajectories of pedestrians within a real-life pedestrian scene. Minor additions to the model have been suggested allowing the detection of incidents through the detection of atypical instantaneous movements and trajectories; the recognition of both simple and complex events by attaching meaning to prototypes representing instantaneous movements and complete trajectories; and trajectory prediction by further attachment of information to prototypes. All the additions mentioned are currently being worked on.

Appendix: Ensuring Correct Distribution of Prototypes

Vector quantisation as described in Section 3.1 has one major problem in that the final distribution of prototypes is extremely sensitive to their initial random placement within the feature space. Prototypes can be 'stranded' in areas where they will never win which will result in a sub-optimal distribution. This is a particular problem in sparse distributions such as those we shall model.

Rumelhart *et al.* [5] propose a method called *leaky learning* where the losing nodes also move their prototypes towards the input vector, but by a much smaller amount. This results in stranded prototypes moving towards the mean of the distribution. For a sparse distribution this is not adequate since the mean of the distribution may itself be 'empty'.

Instead we use a method similar to that suggested by Bienstock *et al.* [9] where each node i has an associated sensitivity. In our implementation, this sensitivity S_i is initially zero and is updated on each epoch

$$\Delta S_i = \begin{cases} -\beta & \text{if } i \text{ winner} \\ \frac{\beta}{k-1} & \text{otherwise} \end{cases} \qquad (10)$$

Where β is in the range (0, 1) and specifies the magnitude of adjustments, and k is the number of prototypes. The value of β should be small relative to the feature space, but large enough to enable stranded nodes to 'escape' within the network's learning period. The form of these updates ensures that for correctly distributed nodes the mean adjustment will be zero.

The sensitivity is subtracted from the Euclidean distance when finding the nearest prototype during learning. In this way a node with +ve sensitivity is more likely and a node with -ve sensitivity is less likely to win the competition. It was found that the use of the sensitivity values also allowed us to train on successive features in a sequence without 'dragging' the nearest prototype along - another prototype is soon forced to win instead. Thus competitive learning with node sensitivities performs a robust vector quantisation.

References

[1] Howarth R. and Buxton H. Analogical representation of spatial events for understanding traffic behaviour. In Neumann B., editor, *10th European Conference on Artificial Intelligence*, pages 785–789. John Wiley & Sons, 1992.

[2] Baumberg A. and Hogg D. An efficient method for contour tracking using active shape models. In *IEEE Workshop on Motion of Non-rigid and Articulated Objects*, pages 194–199. IEEE Computer Society Press, November 1994. IEEE Catalog No. 94TH0671-8.

[3] Cooper D.H. Cootes T.J., Taylor C.J. and Graham J. Training models of shape from sets of examples. In *British Machine Vision Conference*, pages 9–18, September 1992.

[4] Baumberg A. and Hogg. D. Learning flexible models from image sequences. In *European Conference on Computer Vision*, volume 1, pages 299–308, May 1994.

[5] Rumelhart D. and Zipser D. Feature discovery by competitive learning. *Cognitive Science*, (9):75–112, 1985.

[6] Kohonen T. The self-organizing map. *Proceedings Of The IEEE*, 78(9):1464–1480, 1990.

[7] Reiss M. and Taylor G. Storing temporal sequences. *Neural Networks*, 4:773–787, 1991.

[8] Wang D. and Arbib M. Complex temporal sequence learning based on short-term memory. *Proceedings Of The IEEE*, 78(9):1536–1542, 1990.

[9] Cooper L. Bienenstock E. and Munro P. Theory for the development of neuron selectivity; orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, (2):32–48, 1982.