Real Time Deformable Template Tracking

Brendan McCane
Department of Computer Science, University of Otago
Dunedin, New Zealand. email: mccane@cs.otago.ac.nz

Abstract

In this paper, I present a system for real time deformable template tracking that tracks line profiles within regions rather than regions or boundaries. There are two main contributions of this work: the realisation that tracking line profiles rather than areas can be effective and efficient; the extension of the snake minimisation algorithm to allow tree structures. The advantages of the approach are fourfold: since we are matching only a small set of lines within a region the computational load is greatly reduced; deformable region matching is inherent in the method; since boundaries are not being tracked, the problem of background clutter is reduced; tracking of articulated structures is handled naturally.

1 Introduction

The predominant methods of tracking objects in a video sequence have relied on boundary information, usually in the form of active contours of one sort or another [3, 4, 5, 6, 7, 11]. Consequently, significant research effort has been applied to solve the problem of background clutter and occlusion which can confuse a boundary tracking algorithm [7]. More recently, some work has been done on using region information in the form of deformable templates [10, 12] or active meshes [9]. However, few of these techniques claim to work in real time.

Recent research in face recognition [2] indicates that significant identifying features are contained within line profiles of faces. Applying similar techniques to tracking removes the need for using all available region data when matching from one frame to the next and therefore reduces the computational load significantly.

In this paper, I present a system for real time deformable template tracking that is based on the idea of tracking line profiles within regions rather than regions areas. There are two main contributions of this work: the realisation that tracking line profiles rather than areas can be effective and efficient; the extension of the snake minimisation algorithm to allow tree structures rather than simply linear structures. The advantages of this approach are fourfold: since we are matching only a small set of lines within a region, the computational load is greatly reduced; region deformation effectively is a natural side-effect of the method that is proposed; since we are not trying to track boundaries or contours, the problem of background clutter becomes greatly reduced; tracking of articulated structures is handled naturally. In this way we maintain the advantages of both boundary tracking and region tracking - the technique is as efficient as boundary tracking, but maintains the advantages of tracking regions.

2 The Spider Algorithm

The spider tracker forms the heart of the system. A spider is a generalisation of a snake [8, 1] from a 1D linking of features to a 2D linking of features. It is called a spider because it is a tree structure which bears some similarities to a spider with many legs. A tree was chosen for efficiency reasons. A dynamic programming approach to minimisation (the Viterbi algorithm) of the style of Amini et al [1] can be applied to a tree with the same cost complexity per feature as a snake. A single path from the root of the tree to a leaf is equivalent to a single snake, so each node has only a single predecessor, and therefore the dynamic programming approach can be applied in a straightforward manner. The optimal value function of the energy of a spider then, is:

$$S(v_p) = \alpha E_{ext}(v_p) + \sum_{v_c \in C(v_p)} \min_{v_c} \left[S(v_c) + \beta E_{int}(v_c, v_p) \right], \quad (1)$$

where the subscripts p and c indicate parent and child nodes respectively, $C(v_p)$ represents the set of children of node v_p, and the minimum for v_c is calculated over a local neighbourhood for each iteration. So, instead of minimising over a single successor as for a snake, we need to minimise over multiple successors as if we were minimising multiple snakes. Figure 1 shows pictorially what happens for two root node positions in the case of a simplified tree with just three nodes, where each vertex has only two possible new positions. For each possible position of the root node,

we find the minimum energy locations of its children - the energy of the root node in this position is the sum of the minimum energies of the children and the external energy of the root node as indicated in Equation 1. The minimum energy locations of the children are found in a similar way if they are not leaf nodes, or are simply the minimum external energy locations in the case of leaf nodes. In Figure 1, the solid line represents the minimum energy configuration and the dashed lines represent other positions that were tested. This process occurs in a recursive manner so that we calculate the best position of each child node prior to each parent. The minimum energy position of the spider is then calculated by setting the root node to the position of minimum energy and choosing the position of each child node given the position of the root. This change filters down the tree to all leaf nodes. The optimisation algorithm is shown in Algorithm 1 and the spider update algorithm is shown in Algorithm 2. These two algorithms are continuously applied until the spider reaches a local energy minimum.

Algorithm 1 (Spider Optimisation)

```
proc SpiderNodeMinimise(SpiderNode current)
  foreach child ∈ current.children do
    SpiderNodeMinimise(child)
  end
  foreach new_pos ∈ current.neighbours do
    current.energy[new_pos] ← 0
    foreach child ∈ current.children do
      current.energy[new_pos] ←
            current.energy[new_pos]
            +CalculateEnergy(current, child, new_pos)
    end
  end
.
funct CalculateEnergy(parent, child, new_parent_pos)
  min_energy ← inf
  foreach new_child_pos ∈ child.neighbours do
    energy ←child.energy[new_child_pos]+
            InternalEnergy(parent, child,
            new_parent_pos, new_child_pos)+
            ExternalEnergy(parent, child,
            new_parent_pos, new_child_pos)
    if energy < min_energy
      then
        child.best_pos[new_parent_pos] ← new_child_pos
        min_energy ← energy
    fi
  end
  return(min_energy)
.
```

Algorithm 2 (Spider Update)

```
proc UpdateRootPosition(root)
  min_energy ← inf
  foreach new_pos ∈ root.neighbours do
    energy ← root.energy[new_pos]
    if energy < min_energy
      then
        best_pos ← new_pos
        min_energy ← energy
    fi
  end
  foreach child ∈ root.children do
    UpdateChildPosition(child, best_pos)
  end
.
proc UpdateChildPosition(current, parent_pos)
  current.position ← current.best_pos[parent_pos]
  foreach child ∈ current.children do
    UpdateChildPosition(child, current.position)
  end
.
```

As for the case of snakes [1], each iteration has complexity of $O(nm^2)$ for n feature points and m neighbours using a first order internal energy term. This is not true for a more general graph structure whose worst case complexity would be $O(nm^n)$. Hence the reason for restricting the spider to a tree structure.

3 Internal Energy

The internal energy used simply attempts to maintain an equal distance between a spider node and its parent node:

$$E_{int} = \frac{d_{old} - d_{new}}{d_{old}} \tag{2}$$

where E_{int} is the internal energy and d_{old} and d_{new} are the old and new distances between a node and its parent respectively. An angle constraint, similar to the original snake smoothness constraint, can also be employed by either attempting to maintain equal angles for the arc joining two children and their parent, or for the arc joining a grandparent, parent and child, or by maintaining a position and direction at each node and maintaining equal angles between a parent and child. However, all of these solutions require an order of magnitude increase in the efficiency of the method. So, we can either accept somewhat poorer maintenance of shape and improved speed, or good shape maintenance with poorer speed performance. Here, we have decided on the former to satisfy real time constraints.

4 External Energy

Prior to the commencement of tracking, the spider must be initialised in some way. This includes specifying both the structure of the spider (its physical layout), and the contents of the spider. In this case, the structure is specified and a profile of colour values is learned from a single example image. This profile is learned simply by sampling the image at unit pixel increments along the legs of the spider.

The external energy is calculated as the sum of squared differences between the stored profile and the current frame:

$$E_{ext} = \left(1.0/N \sum_{i=0}^{N} (p_i - I_i)^2 \right)^{0.5} \quad (3)$$

where N is the length of the profile, p_i is the colour of the profile at point i and I_i is the colour of the image at a point i/N parts along from the start point to the end point (parent position to the child position).

5 Results

The spider tracker has been successfully tested on several image sequences. Here we show two to demonstrate the strengths of the tracker.

The spider tracker is ideal for real time applications such as eye-tracking. Figure 2 shows three frames of a 130 frame sequence with the resultant spider painted onto the image in white (obtained from the Signal Analysis and Machine Perception Laboratory at Ohio State University, `http://sampl.eng.ohio-state.edu/~sampl/`). Each frame is 360×240 pixels in size (the frames are clipped in the figure) and although they are shown here in greyscale only, the source images are actually colour. The centre of each radial wheel represents the spider's best estimate of the position of the centre of the eye. This is a high quality image sequence, with quite small motion between frames, and hence the tracker has been extremely successful. Execution speeds of 9.5ms/frame or 100 frames/second were achieved (Pentium III, 800MHz, Linux) not counting image acquisition time. With such high quality data, the pupils were correctly tracked across the sequence, so the technique could also be used for estimating gaze direction.

Figure 3 shows three frames of a much poorer quality 80 frame sequence. Each frame is 160×120 pixels in size and although they are shown here in greyscale only, the source images are actually colour. This example was chosen to highlight the tracker's ability to track articulated structures and its relative insensitivity to background clutter. Execution speed on this sequence was 10.5ms/frame. This sequence has not been tracked 100% successfully as the node attached to the right hand drifts down the forearm. This is due to the uniformity of the colour along the arm. A similar problem is not realised in the left hand due to the watchband holding that node in place.

6 Conclusion

In this paper, I have presented a new method for performing deformable template matching that runs in real time and have demonstrated the efficacy of the method on tracking eyes and articulated structures. For the two image sequences presented here, the execution speed was approximately 10ms/frame. The speed of the technique is dependent on the complexity of the spider itself (number of nodes and the number of neighbourhood positions to search) and the size of the connectors between the nodes.

Although the results are promising, the technique does have several limitations. Firstly, like many tracking techniques, it does not deal with occlusion events particularly well. Further, if part of the target undergoes large inter-frame motion, then that part of the target will likely be lost by the tracker. Finally, as with many vision techniques, there are several parameters which need to be set and this is typically necessary on a per-sequence basis. Areas for further work include extending the technique to handle partial occlusion and developing techniques to learn the parameter weights during execution.

References

[1] Amir A. Amini, Terry E. Weymouth, and Ramesh C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9 (September)):855–867, 1990.

[2] Olivier de Vel and Stefan Aeberhard. Line-based face recognition under varying pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1081–1088, 1999.

[3] P. Delagnes, J. Benois, and D. Barba. Active contours approach to object tracking in image sequences with complex background. *Pattern Recognition Letters*, 16(February):171–178, 1995.

[4] S. Dickinson, P. Jasiobedzki, H. Christensen, and G Olofsson. Qualitative tracking of 3d objects using active contour networks. In *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 812–817, Seattle, 1994 June. IEEE Computer Society Press.

[5] Davi Geiger, Alok Gupta, A Costa, and John Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(3):294–303, 1995.

[6] K Hata, J Ohya, F Kishino, and R Nakatsu. Automatic extraction and tracking of contours. *IEEE Proceedings of ICPR '96*, pages 441–445, 1996.

[7] Michael Isard and Andrew Blake. Condensation–conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.

[8] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. In *International Journal of Computer Vision*, pages 321–331, 1988.

[9] D. Molloy and P.F.Whelan. Active-meshes. *Pattern Recognition Letters*, 21:1071–1080, 2000.

[10] Nikos Paragios and Rachid Deriche. Unifying boundary and region-based information for geodesic active tracking. In *Computer Vision and Pattern Recognition*, volume 2, pages 300–305, Fort Collins, Colorado, 1999. IEEE Computer Society.

[11] Natan Peterfreund. Robust tracking of position and velocity with kalman snakes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):564–569, 1999.

[12] Y. Zhong, A.K. Jain, and M-P. Dubuisson-Jolly. Object tracking using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, 22(5):544–549, 2000.

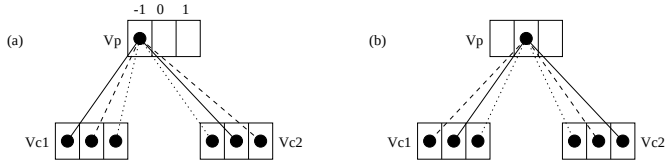

Figure 1: Finding the locally optimal position of the spider.

(a) Frame 0 (b) Frame 81 (c) Frame 109

Figure 2: The mom sequence. Notice how the spiders continue tracking successfully despite considerable deformation of the eyes on the picture plane.

(a) Frame 0 (b) Frame 16 (c) Frame 48

Figure 3: An articulated sequence. The spider still successfully tracked the structure despite the articulation and background clutter.