Full-3D Edge Tracking with a Particle Filter

Georg Klein and David Murray
Department of Engineering Science, University of Oxford, UK
{gk,dwm}@robots.ox.ac.uk

Abstract

This paper demonstrates a real-time, full-3D edge tracker based on a particle filter. In contrast to previous methods this system is capable of tracking complex self-occluding three-dimensional structures. The system exploits graphics hardware in a novel manner, allowing it not only to perform hidden line removal for each particle but also to evaluate pose likelihoods directly on the graphics card. This approach allows video-rate filtering with hundreds of particles on a standard workstation.

1 Introduction

Tracking the pose of a camera relative to a known object in real-time is a common computer vision task with a variety of applications from manufacturing to augmented reality. Since the early work of Harris [4] this task has often been accomplished by observing edges in the image: using a CAD model of the object to be tracked, the camera pose which best aligns rendered model edges to detected image edges can be determined. Edges are easy to detect in images, offer a large degree of invariance to pose and illumination changes and have some resilience to difficult imaging conditions (noise and blur) and so the tracking of edge models remains an active field of computer vision today.

The primary disadvantage of edges as features to track is that one image edge looks much like another. Whereas a rich selection of description techniques exist for matching point features, edges are often matched simply by image proximity to a prior. Without a valid prior pose estimate, most edge-based systems will break and not recover track; Typically this means that every single frame of a sequence must be correctly tracked to provide the prior estimate for the next frame. Much recent work on edge-based tracking has therefore attempted to increase frame-to-frame robustness to such a high level that usefully long sequences can be tracked.

Approaches to increase frame-to-frame robustness have included the use of robust estimation techniques [1, 3], the addition of external sensors [7], and the use of point features for initialisation [17, 15]; Further, attempts at considering multiple edge correspondence hypotheses have shown great potential [6]. However, all of these methods are fundamentally uni-modal in that they calculate a single (two for [15]) Gaussian posterior pose for each frame, which then provides a single prior pose for the next frame; if the estimate is sufficiently incorrect, tracking will fail.

Particle filters provide an alternative approach to propagating pose estimates. Posterior and prior distributions are no longer limited to single Gaussians but can adopt truly non-Gaussian, multi-modal forms as convincingly demonstrated by Isard and Blake's CONDENSATION [5] algorithm. Despite the algorithm's impressive performance in

real-time 2D edge tracking, an extension to 3D edge models has been conspicuous in its absence until very recently: Pupilli and Calway [12], in an extension of their earlier work using point matches [11], have shown that full 3D tracking using edge models and an annealed particle filter is fundamentally possible and can indeed provide the robustness advantages already demonstrated by CONDENSATION.

This paper presents an alternative implementation of full 3D particle-filter-based edge tracking which addresses the primary limitation of the system proposed in [12], which is restricted to tracking very simple 3D objects. The challenge lies in performing hidden edge removal: whereas single-hypothesis trackers can afford to perform computationally expensive rendering only once per frame, a particle filter requires this rendering step to be performed *for each particle.* Pupilli and Calway do not attempt hidden edge removal and are therefore restricted to tracking objects which do not self-occlude.

Our approach removes this limitation. By utilising the graphics acceleration hardware commonplace in today's workstations, we are able to perform hidden line removal independently for each particle. However merely using hardware acceleration to render visible edges does not solve the problem, since transferring the visible edges of each particle back to the CPU is impractically slow. Instead, we show that it is possible to measure each particle's likelihood directly on the graphics card. Depending on model complexity, this can be done at rates in excess of 10,000 pose hypotheses per second, and allows our particle filter to track objects of a complexity comparable to that supported by state-of-the-art unimodal systems.

Experimental results show a resilience to under-constrained scenes and erratic motion typical of particle-filter-based systems. Further, the particle filter is highly flexible in terms of motion models supported; this allows the correct integration of a fast motion estimation algorithm [8] which produces very noisy rotation estimates. The combined system is able to correctly track sequences not previously trackable without additional inertial sensors.

2 Background

Most single-hypothesis edge tracking systems operate in a manner similar to that of Harris' RAPiD system [4] already described in the previous section: Edges are rendered based on the prior pose estimate, and sample points are initialised at regular intervals along the visible edges. Perpendicular 1D searches locate the nearest image edge and these distances are measured. From these distances and a motion Jacobian, the pose adjustment which minimises the sum-squared edgel distances is found.

Standard least-squares is sensitive to outlier measurements and so later approaches have attempted to reduce their sensitivity to outliers by adopting a form of robust estimation. [1] achieves this using RANSAC to estimate individual edges and case deletion to estimate pose. [6] goes beyond this by constructing a multi-modal likelihood estimate from RANSAC at the low level and an exhaustive search at high level. A common alternative to RANSAC is the use of a robust M-estimator [3, 9, 7, 17]. This replaces the square-error metric with one which is less sensitive to outliers and typically requires multiple iterations of re-weighted least squares to converge.

A further method of increasing edge tracking robustness is the use of some form of pose initialisation. Any estimate of camera motion between frames helps edge tracking

avoid correspondence errors. Physical rate gyroscopes are used in [7] to obtain the prior pose; the system also modifies edge detection behaviour to account for motion blur. A software approach is taken by [15, 17]: in these systems, edge tracking is initialised by matching interest points from the previous frame. Highly impressive tracking results are presented but it is not clear that these systems would operate correctly under motion blur.

The use of graphics hardware to facilitate computer vision tasks has been commonplace for many years. For example, while the visibility of sample points is determined using software (a view-sphere) in [4], the OpenGL and stencil buffering is used for the same task in [3]. The use of texture mapping to remove radial distortion from camera images has been proposed in [18] (and is also used here). However, most early examples use the graphics system for some sort of graphical task, whereas now the advent of programmable pixel and vertex shaders (along with the massive bandwidth of graphics processors) has generated interest in the using GPUs for general-purpose computation.

Previous work most closely related to this paper is that of Rao and Hodges[13] who attempt to accelerate markerless human motion capture. Like here, a particle filter is used to represent the tracking system's state, and a method of accelerating the likelihood calculation of the filter is developed; this is however where the similarities end. The measurements for each particle are not independent, and hence a breadth-first evaluation of particles is performed; state for each particle is stored in large look-up textures which store the accumulated information about each particle. These look-up textures are alternated as inputs and outputs and iteratively filled before finally being read back to the CPU. (By contrast, all the information passed from GPU to CPU in our approach is passed back by an out-of-band extension, as described in Section 3.2.) The authors report an impressive $20\times$ speed-up moving from a 3.06GHz CPU to a GeForce 5900FX GPU.

3 Method

3.1 Particle filtering

As in [12], the proposed system is fundamentally an implementation of the CONDENSATION algorithm with annealing. The system differs from previous approaches primarily in the choice of motion model and observation function. The algorithm attempts to process live 640×480 video at 30Hz.

We represent camera pose as a 4×4 transformation matrix X which transforms points from homogeneous world coordinates to the camera coordinate frame:

$$\boldsymbol{p}_{camera} = X\boldsymbol{p}_{world}, \qquad X = \begin{bmatrix} R & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{1}$$

The posterior distribtuion (denoted $+$) at frame $t-1$ is represented by a set of N particles $\{X_1^+ ... X_N^+\}$ and associated weights $\{w_1 ... w_N\}$. For the next frame t, samples are drawn from this distribution and perturbed by a motion model to form the frame's prior distribution, which is a set of pose N particles $\{X_1^- ... X_N^-\}$ with uniform weight:

$$X_i^- = \text{Perturb}(X_n^+) \tag{2}$$

where n is sampled from $1..N$ according to the posterior particles' weights $\{w_1 ... w_N\}$, and Perturb() adjusts camera pose according to a motion sampled from a statistical motion model (discussed in Section 3.4.)

Next, each of the prior pose particles is evaluated against the received video frame. The known 3D object to be tracked is rendered according to the particle's pose and the resulting edges are compared with edges found in the video image to produce a measure of the particle's likelihood; the particles weighted by this likelihood then form the posterior distribution for frame t:

$$X_i^+ = X_i^-, \qquad w_i = \text{Likelihood}(X_i^-) \tag{3}$$

In practice we perform the above procedure twice per frame with different motion and likelihood models to produce an annealed filter [2, 11, 12] where each stage also uses a different number of particles, however the system is in principle a straightforward particle filter. The challenge lies in evaluating the likelihood for each particle fast enough to permit real-time operation, and this procedure is described next.

3.2 Rendering and Likelihood Evaluation

Previous real-time, edge-based particle filters have tracked planar contours or simple 3D objects for which self-occlusion can be ignored. Here we consider complex 3D models which require that hidden edges be removed. If one assumed that the particle distribution is not too broad and does not span significant aspect changes, one might be able to determine edge or sample-point visibility once for the mean pose and use this approximation for all particles. This would however sacrifice some of the flexibility of particle filters (in particular the ability to track multiple distinct modes) and so our approach treats each particle individually and no common rendering or linearisation is performed.

Assuming the system performs around 300 likelihood measurements per frame and operates at 30Hz, this requires that rendering and likelihood evaluation take no more than around $100\mu s$. To our knowledge this is not possible in software on standard workstations. We therefore exploit the hardware graphics acceleration which comes as standard with most computers available today.

Rendering an edge-based model with hidden lines removed is easily done in OpenGL using the depth buffer, which stores scene depth for each pixel. For each particle, the tracked object's faces are rendered into the depth buffer using a pin-hole projection model and the particle's pose matrix. Next, the object's edges are drawn with depth-testing enabled: Every edge pixel is only drawn if its depth is not greater than that already stored in the depth buffer. All of these operations are highly optimised on graphics cards and this rendering step can be performed at rates well in excess of 10kHz.

Unfortunately, while performing hidden edge removal at high speeds is straightforward, accessing the results with the CPU is not. Graphics cards are designed for output to screen and not for transferring data back to the CPU; while pixel values can be accessed using `glReadPixels()` this incurs massive performance penalties as it is incompatible with the asynchronous streaming operation of today's graphics cards. The solution to this problem is to calculate each particle's likelihood directly on the graphics cards. This minimises the amount of information which must be transferred from graphics card back to the CPU: here, two integer values per particle suffice. Further, by using an OpenGL extension specifically designed for asynchronous feedback, pipeline stalls associated with explicit pixel read commands can be avoided.

The `GL_OCCLUSION_QUERY` extension [14] exposes an array of counters which can be used to obtain a count of how many pixels from a drawing operation pass the depth and

stencil tests. For each particle, we employ two counters to obtain two pixel quantities: firstly, the number v of pixels which make up the visible edges of the model; secondly, the number d of these edge pixels which also coincide with edges detected in the video image. This is done first rendering the model's faces into the depth buffer, and then by rendering the model's edges twice: the first pass uses the standard OpenGL pipeline, whereas the second pass employs a pixel shader program which only draws pixels if they lie on or close to a video frame edge. A description of this pixel shader and of the preparatory steps required for its operation follows in Section 3.3. From the two pixel counts, the particle's likelihood (and posterior weight) is found:

$$\text{Likelihood}(X_i^-) \propto \exp\left(k\frac{d_i}{v_i}\right). \tag{4}$$

The above formulation is similar to Pupilli and Calway's inlier/outlier test, except that here it is applied to *every single edge pixel.* A complication is presented by the fact that a different number of tests (v) is performed for each particle: this is unavoidable since each particle may make the object appear at different sizes on screen, and may reveal or hide a different number of edges. If the exponential were simply $\exp(-(v-d))$ this would bias the filter towards those particles which render as few pixels as possible, and here this is avoided by division through v. The tuning constant k controls the discrimination of the filter, with high values boosting the relative weight of strong particles.

3.3 Pixel shader and frame preprocessing

Pixel shaders are programs which can be executed on the graphics card to modify some properties of pixels being rendered by the graphics card. The OpenGL specification imposes severe limitations on the capabilities of these programs in order to ensure high execution speed, which restricts their use for general-purpose computation; they are however entirely appropriate for the task here.

To produce the count of actual-video-edge pixels d, a pixel shader is activated which tests each incoming edge pixel to check if its location matches an edge detected in the video image. In its simplest form, the pixel shader could simply calculate the gradient intensity in the video image (which is available as a texture on the graphics card) at the pixel's location and pass the pixel if this exceeds a threshold, however this would be slow and produce very narrow likelihood peaks; instead, the shader makes use of a pre-generated texture map which contains, for each pixel, two items of information: the distance to the nearest video edge and the direction of this edge. The pixel shader checks for suitable proximity to an edge, and then compares its direction to that of the edge being rendered.[1] If no edges are nearby or the directions do not match, the pixel is discarded and will not increment the pixel count.

The texture map required for these lookups is generated afresh for every incoming video frame. This processing involves a number of steps:

1. Rectification. OpenGL supports standard pin-hole rendering but no radial distortion; hence the incoming video image is warped to remove the calibrated camera's radial distortion. This is done using a textured 20×20 grid.

[1] OpenGL note: Pixel shaders have no access to primitive information and so the direction of the edge being rendered must be passed along by a suitably written vertex shader, which in turn requires 3D line orientation to be passed along with vertex data.

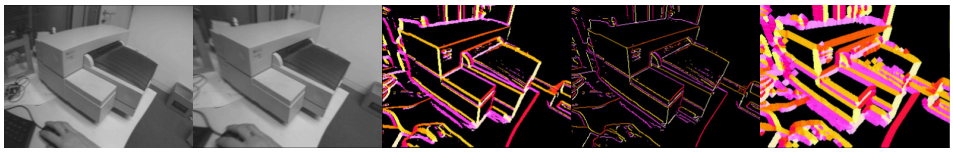


Figure 1: Frame preprocessing. From left to right: Raw frame; undistorted frame; thresholded Sobel image; thinned edge image; spread image. Low-res versions shown here.

2. Sobel transform. The image gradients G_x and G_y for each pixel are calculated. These are transformed into gradient magnitude an gradient direction, which is stored as a normalised 2-vector. Pixels whose gradient magnitude exceeds a threshold are marked as being edge pixels.

3. Non-maximal suppression (Thinning.) Edge pixels whose gradient magnitude is not higher than that of their two neighbours in the direction of the image gradient are eliminated.

4. Distance transform (Spreading.) Pixels which are not themselves edges adopt a distance metric and the gradient direction of any adjacent edge pixel.

All of these steps can be performed on the graphics card using pixel shaders. The result of the individual steps is illustrated in Figure 1. The last step of the procedure can be performed multiple times; each pass spreads edges by one pixel and therefore broadens the peaks of the likelihood function.

3.4 Motion model

Each prior particle X^- is formed by perturbing a posterior pose by a sample from a statistical motion model. A simple motion model draws these motions from a Gaussian noise model:

$$\text{Perturb}(X^+) = MX^+ \tag{5}$$

$$\text{with} \quad M = \exp(\boldsymbol{\mu}), \quad \boldsymbol{\mu} \sim N(\mathbf{0}, \sigma^2 I_6) \tag{6}$$

where M is a 4×4 transformation matrix representing camera motion, generated by the exponential map[2] from a normally-distributed motion six-vector $\boldsymbol{\mu}$. Here σ, the motion standard deviation, can be tuned to the specific application; depending on the scene to be tracked, a different value may be used for the rotation and translation components.

This motion model is less general than the uniform distribution proposed in [12] but we find it yields better results. Tracking accuracy for slowly-moving or stationary objects can further be increased by sampling motions from a two-part Gaussian mixture, where a small fraction of the motion samples are drawn from a narrower Gaussian with one-tenth the standard deviation.

The motion model just described does not estimate camera velocity. Compared to the constant-velocity models frequently used in unimodal tracking systems, this allows the system to tolerate far more erratic acceleration (shake) but limits the maximum trackable

[2] This framework which represents the group SE(3) is explained in any of [3, 6, 7, 15].

velocity. While it is possible to maintain a per-particle velocity model to handle larger velocities, we here use the visual gyroscope algorithm of Klein and Drummond [8] to estimate camera rotation instead. This algorithm attempts estimate camera rotation from the structure of motion blur present in a video image, with low computational overhead (2-3ms). The algorithm has several limitations and makes many simplifying assumptions, with the result that motion estimates are very noisy; however, this behaviour can be modeled and so integrates well with the particle filter framework.

In general, the algorithm is better at estimating the camera's axis of rotation than the magnitude of this rotation. We do not therefore adjust the algorithm's estimate of the rotational axis, but adjust the motion magnitude for each sample. If the original magnitude estimate is ω, each particle samples an adjusted version from one of three classes:

$$\omega' = \begin{cases} \omega + \Delta \\ 0 \\ -\omega + \Delta \end{cases} \qquad \Delta \sim N(0, \sigma_b) \tag{7}$$

The majority of samples are drawn from the first class which corresponds to the algorithm working correctly, albeit noisily; the second class foresees occasional completely erroneous measurements; and the class arises due to the algorithm's inherent forwards-backwards ambiguity. The ambiguity is usually resolved by testing the sum-squared difference of patches sampled from the previous frame, but this procedure is imperfect so a small number of samples are projected "backwards" to account for this.

A transformation matrix M_b resulting from the sampled blur gyroscope estimate is left-multiplied to the normally-generated (albeit with lower rotational noise) random motion to compete the system's motion model.

3.5 Implementation notes

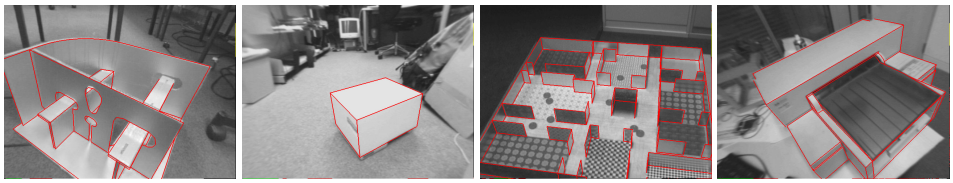
This section describes some details relating to the procedures described above.

Annealing: In a crude version of annealed particle filtering [2, 11, 12], we operate two tracking stages per frame. The first stage corresponds to a broad search using many particles and a blurred likelihood function, and the second iteration uses fewer particles, smaller motions and narrower likelihood peaks to refine the estimates from the first stage. The likelihoods peaks are adjusted by spreading the detected edges to different widths.

Minimal CPU - GPU data transfer: The geometry of the model to be tracked can be cached on the graphics card. The only data which the CPU has to send the to the graphics card each is each incoming video frame and the particles' transformation matrices. The only data received back are the edge pixel counts.

Sub-sampled distance map: The distance transform/spreading operation requires many pixel accesses and is time consuming, particularly for the first iteration of tracking where the likelihood peaks should be substantially blurred. A sub-sampled (320×240) map is therefore used for the first iteration since this is far faster to generate. The second iteration operates on a full-resolution map.

Sub-sampled depth buffer: Adjacent pixels along an image edge are so closely correlated that testing each individual edge pixel is redundant. For single-hypothesis trackers, it is common to spread sample points a distance of 10-20 pixels apart along an edge. Sampling only every *n*th edge pixel also reduces the graphics bandwidth required and so only every 4th pixel is sampled. Instead of explicitly drawing stippled lines, this is



Ship: 280+100 Box: 620+100 Maze: 250+100 Printer: 370+100

Figure 2: Tracked objects and the maximum number of particles for 30Hz operation

here achieved by using a sub-sampled depth buffer (160×120) since this further achieves a bandwidth reduction for clearing and populating the depth buffer. However, this also means that hidden line removal can be inaccurate to approximately four pixels. Apart from this, the accuracy of the system is unaffected.

Parallel processing: Due to the asynchronous graphics it is possible to perform processing on the CPU while waiting for results from the GPU. For example, the frame's motion blur is analysed on the CPU while the GPU is pre-processing the incoming frame.

4 Results

The system has been implemented on a P4 3.2Ghz with an nVidia GeForce 6800 (12 pipeline, AGP) graphics card. Of the 33ms/frame budget, 7ms are required by the pre-processing described in Section 3.3 and the remaining graphics time is used for likelihood evaluations. To maintain a 30Hz operating rate, the number of pose particles tested for the first annealing stage is automatically adjusted, while the second stage uses 100 particles.

A variety of 3D models have been tracked, using both live video and prerecorded sequences. Since the time required for likelihood evaluation varies with model complexity, the number of particles used changes. Figure 2 shows some of the models tracked and the number of particles supported at 30Hz. As ever, it is difficult to convey tracking performance in pictures and numbers, so an illustrative video file has been included; some frames from this are shown in Figure 3.

Tracking robustness for simple objects such as the box is good. Compared to unimodal edge-based systems, resilience to abrupt camera motions is very high, and local minima around aspect changes do not cause problems. However, tracking accuracy is not as high as systems which perform direct pose optimisation. Further, it is not clear how to extract a single pose estimate to render from the posterior distribution; the mode pose exhibits much jitter, while the mean pose can lag behind rapid motions. This is especially apparent in the printer video, which shows the mean pose; very often this is misaligned, since the sequence continually contains erratic motion which causes the filter to spread out. The printer video further often shows the system tracking an incorrect local minimum, however the filter generally re-converges on the correct pose quickly.

The system was tested on a pre-recorded sequence of a ship part in which a camera undergoes rapid rotation with a slow shutter speed. The challenge here is tracking in the continual presence of substantial motion blur. This sequence has previously been successfully tracked through the use of rate gyroscopes to predict camera rotation and motion blur [7]. Here this sequence is trackable without gyroscopes using rotation estimates calculated from the motion blur, despite the fact that edge detection is not adjusted for blur

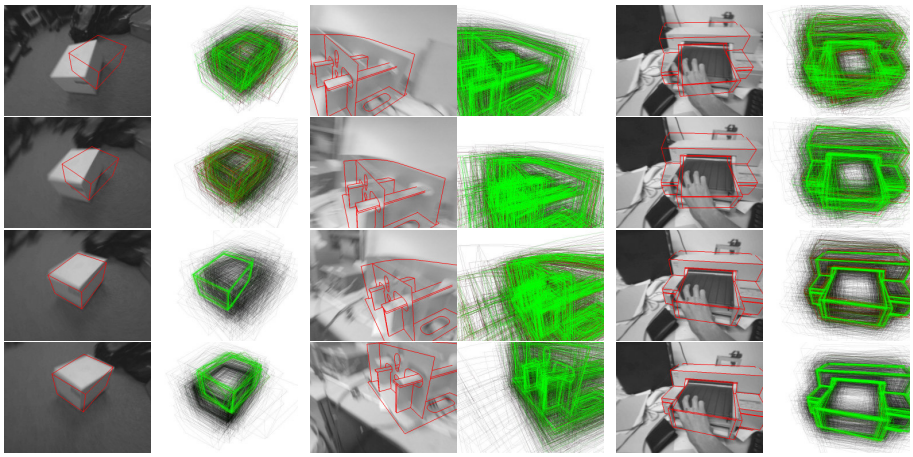


Figure 3: Selected frames from tracked sequences with rendered posterior distributions; the first iteration is shaded gray to green, the second red. On the left, a single correctly-aligned particle allows the filter to converge on the correct pose. Center, the posterior diverges along the direction of motion blur as the camera pans rapidly. The filter re-converges when the blur eases. Right, the filter recovers from a misaligned pose.

in any way. Since the rotation estimates produced by the visual gyroscope are sometimes erroneous, a single-hypothesis system fails to track this sequence.

A further comparative test was performed on a sequence of the maze model tracked successfully in [6]. This difficult sequence in a highly textured environment with many parallel edges was not trackable at 30Hz. Even if the number of particles is raised to 500, tracking remains fragile and tends to fail. The poor performance is presumably partially due to the camera motion, which is smooth but rapid (and thus suited for a constant velocity model rather than the one used here); further, [6] uses a texture change-point edge detector rather than the high intensity gradient model used here.

5 Conclusions

This paper has demonstrated a particle-filter-based edge tracker capable of tracking complex 3D objects with self-occlusions. Tracking can be performed at video rate by exploiting hardware acceleration to perform hidden line removal and likelihood calculations.

The particle filter has robustness advantages over previous systems, particularly when exposed to rapid, unpredictable accelerations. Further, the flexibility of the particle filter allows the integration of a wide range of motion models, which is exploited here by utilising a fast but noisy visual gyroscope. A disadvantage of the proposed system is increased jitter in stationary scenes. Further, the simple likelihood model dictated by real-time constraints makes the integration of more advanced edge detection [16, 6] difficult.

Future work will attempt to improve the pose estimates selected for rendering, since neither mean nor mode of the posterior are satisfactory. Further, accuracy improvements could be likely obtained by performing per-particle optimisation (cf. FastSLAM 2.0 [10]) with the edge models. This is unlikely to be possible in real-time on current hardware but may well be feasible in the near future.

Acknowledgement This work was supported by EPSRC grant GR/S97774/01.

References

[1] M. Armstrong and A. Zisserman. Robust object tracking. In *Proc. Asian Conference on Computer Vision*, volume I, pages 58–61, 1995.

[2] J. Deutscher, A. Blake, and I. Reid. Articulated Body Motion Capture by Annealed Particle Filtering. In *Proc. Intl. Conference on Computer Vision and Pattern Recognition*, volume 2, pages 126–133, 2000.

[3] T. Drummond and R. Cipolla. Real-time tracking of complex structures with on-line camera calibration. In *Proc. British Machine Vision Conference (BMVC'99)*, volume 2, pages 574–583, Nottingham, September 1999. BMVA.

[4] C. Harris. Tracking with rigid models. In A. Blake, editor, *Active Vision*, chapter 4, pages 59–73. MIT Press, 1992.

[5] M. Isard and A. Blake. CONDENSATION - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[6] C. Kemp and T. Drummond. Dynamic measurement clustering to aid real time tracking. In *Proc. 10th IEEE International Conference on Computer Vision (ICCV'05)*, volume 2, pages 1500–1507, Beijing, November 2005.

[7] G. Klein and T. Drummond. Tightly integrated sensor fusion for robust visual tracking. In *Proc. British Machine Vision Conference (BMVC'02)*, volume 2, pages 787 –796, Cardiff, September 2002. BMVA.

[8] G. Klein and T. Drummond. A single-frame visual gyroscope. In *Proc. British Machine Vision Conference (BMVC'05)*, volume 2, pages 529–538, Oxford, September 2005. BMVA.

[9] E. Marchand, P. Bouthemy, F. Chaumette, and V. Moreau. Robust real-time visual tracking using a 2D-3D model-based approach. In *Proc. 7th IEEE International Conference on Computer Vision (ICCV'99)*, volume 1, pages 262–268, Kerkyra, Greece, September 1999.

[10] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. 16th Intl. Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, 2003. IJCAI.

[11] M. Pupilli and A. Calway. Real-time camera tracking using a particle filter. In *Proc. British Machine Vision Conference (BMVC'05)*, pages 519–528, Oxford, September 2005. BMVA.

[12] M. Pupilli and A. Calway. Real-time camera tracking using known 3D models and a particle filter. In *Intl. Conference on Pattern Recognition*, August 2006.

[13] S. Rao and L.F. Hodges. Interactive marker-less tracking of human limbs. *Submitted to Transactions on Visualization and Computer Graphics*, 2005.

[14] A. Rege. Occlusion (HP and NV extensions). Game Developer's Conference 2002, http://developer.nvidia.com/object/gdc_occlusion.html, accessed April 2006.

[15] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. 10th IEEE International Conference on Computer Vision (ICCV'05)*, volume 2, pages 1508–1515, Beijing, November 2005.

[16] A. Shahrokni, T. Drummond, and P. Fua. Texture boundary detection for real-time tracking. In *Proc. 8th European Conference on Computer Vision (ECCV'04)*, volume 3022, pages 566–577, Prague, May 2004.

[17] L. Vacchetti, V. Lepetit, and P. Fua. Combining edge and texture information for real-time accurate 3D camera tracking. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pages 48–57, Arlington, VA, November 2004.

[18] B. Watson and F. Hodges. Using texture maps to correct for optical distortion in head-mounted displays. In *Proc. IEEE Virtual Reality Annual Symposium*, pages 172–178, March 1995.