PWP3D: Real-time segmentation and tracking of 3D objects

Victor A. Prisacariu
victor@robots.ox.ac.uk
Department of Engineering Science, University of Oxford

Ian D. Reid
ian@robots.ox.ac.uk
Department of Engineering Science, University of Oxford

Abstract

We formulate a probabilistic framework for simultaneous 2D segmentation and 2D–3D pose tracking, using a known 3D model (of arbitrary shape) of the segmented object. Our technique is region-based; at each frame we maximise the discrimination between statistical foreground and background models, by adjusting the pose parameters iteratively. Unlike all previous work in 3D tracking, we use posterior membership probabilities for foreground and background pixels, rather than pixel likelihoods, and during periods of stable tracking we allow adaptation of the statistical foreground and background models. We support our ideas with a real-time implementation, and use this to generate experimental results on both real and artificial video sequences, with a number of 3D models, to showcase the qualities of our tracker, and to demonstrate the benefit of using pixel-wise posteriors rather than likelihoods.

1 Introduction

Fast and accurate image segmentation and pose tracking are two fundamental tasks in computer vision. Many current studies treat these two tasks independently and there are few articles which study the two tasks simultaneously and even fewer which consider the problem of real time performance. In the current work, we develop a method based on the assumption that, given an accurate 3D model of an object, its segmentation from any given image is fully defined by its pose. Our method allows for fast 2D–3D pose tracking and 2D segmentation using a single, unified, energy function.

Inspired by [1], (which considered *2D* tracking and segmentation), we aim to maximise the posterior per-pixel probability of foreground and background membership as a function of pose. This was shown to yield a better behaved energy function (in the 2D case), when compared to a standard level set formulation such as [2]. In our case we assume a known 3D model and calibrated camera, and seek the six degree of freedom rigid transformation that maximises the pixel-wise posterior energy function. Like [1] we represent the region statistics by colour histograms and adapt these online. Our method is reduced to a per-pixel computation which is very easily parallelisable, so we are able to achieve real time performance by developing massively parallel algorithms which we implemented using the NVIDIA CUDA framework. The minimization is done using gradient descent (though other more sophisticated minimisation techniques are not precluded).

BMVC 2009 doi:10.5244/C.23.47

The most closely related work to our own, likewise is based on simultaneous segmentation and tracking using region-based criteria [3, 7, 8]. In [7] the authors evolve an infinite dimensional active contour in two iterative steps: first the contour, represented by a zero level-set of a 3D embedding function, is evolved to find a segmentation, in the expectation that the contour will then match the projection of the occluding contour of the 3D object. Second, each point on the contour is back-projected to a ray (represented in Plücker coordinates), and the pose sought that best satisfies the tangency constraints that exist between the 3D object and these rays. In some respects it bears similarity to Harris' seminal work [5], with the level set evolution to find the extremal contour replacing Harris' 1D searches for edges. This two-stage method places no constraints on the shape of the contour, even though in reality the only valid contours are the ones which are the projection of the occluding boundary on the object, and will therefore fail if the first stage fails to obtain a very good segmentation. Furthermore our experience with the two-stage method is that the 2D-3D pose matching did not have a large basin of convergence.

A cleaner approach is proposed in [8]. Here, the unconstrained contour evolution stage is removed, and the minimisation takes place by evolving the contour (approximately) directly from the 3D pose parameters. The direction of contour evolution is determined by the relative foreground/background membership probabilities of each point, while the amount of evolution is apparently a "tunable" parameter. The pose is then determined as in the method of [7].

The work most closely related to ours, [3], also proposes a direct minimisation over the pose parameters. It is based on an energy function summing two integrals – one over the foreground, one over the background. Differentiating these integrals with respect to the pose parameters is achieved using the Leibniz-Reynolds transport theorem, yielding an integral along the occluding contour and two surface integrals. These surface integrals collapse to zero because the authors choose to represent the statistics of the two regions of the image with the functions used in [9].

In our work we adopt the pixel-wise posterior background/foreground membership approach of [1] (which would not yield a convenient collapse of these terms), but rather than formulate our energy over the sum of separate integrals for foreground and background, we use the Heaviside step function, evaluated on the 3D embedding function of the contour, to delineate foreground and background in the integral. This has two significant advantages: (i) the maths is simpler, and permits greater flexibility in the choice of optimisation method (for example methods which use higher than first-order derivatives); and (ii) more importantly, it is trivial to replace the Heaviside step function with a blurred version, and in consequence we can very naturally allow for a degree of uncertainty in the location of the contour.

The remainder of the paper is structured as follows: we begin by briefly describing the geometry and setting up our notational conventions. In Section 3 we describe the mathematical foundations and in Section 4 we derive our algorithm and discuss its implementation. In Section 5 we consider various qualitative and quantitative evaluations of our method on both real and synthetic data, and we conclude in Section 6.

2 Notation

Let $\mathbf{X_0} = [X_0, Y_0, Z_0]^T \in \mathbb{R}^3$ be a 3D point in the object coordinate frame, while $\mathbf{X} = [X, Y, Z]^T = \mathbf{R}\mathbf{X_0} + \mathbf{T} \in \mathbb{R}^3$ is a point in camera coordinate frame. The image itself is denoted $\mathbf{I}$, and the image domain by, $\Omega \subset \mathbb{R}^2$, with an area element denoted $d\Omega$. An image pixel $\mathbf{x} = [x, y]$ in

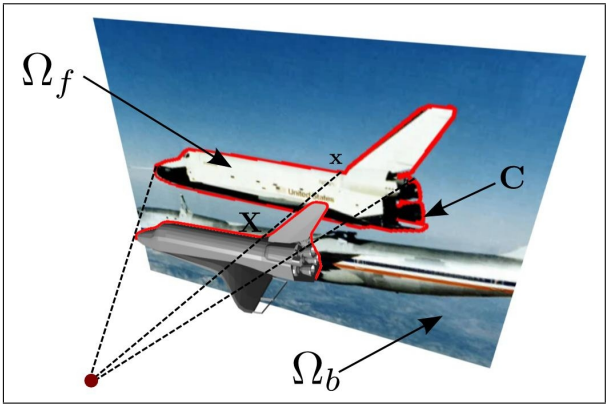


Figure 1: Representation of the object showing: the contour of the projection **C** and the corresponding contour around the visible part of the 3D model, the foreground region Ω_f and the background region Ω_b, a 2D point **x** on the contour and its corresponding 3D point in the object coordinate frame **X**

this domain has the value $\mathbf{I}(\mathbf{x}) = \mathbf{y}$ (in our experiments this is an RGB value).

We assume a calibrated camera, so without loss of generality, $x = X/Z, y = Y/Z$. The pose parameters – i.e. rotation and translation relating points $\mathbf{X_0}$ in the object frame to the camera frame – are denoted by λ_i.

The projection of the occluding contour of the object is denoted by the curve **C**. This closed curve is the zero level-set of an embedding function $\Phi(\mathbf{x})$ [2]. The curve **C** segments the image into disjoint regions Ω_f and Ω_b, for foreground and background respectively. Each region has its own statistical appearance model, $P(\mathbf{y}|M)$ where M is one of M_f or M_b.

Finally, by $H_e(x)$ we denote the smoothed Heaviside step function and by $\delta_e(x)$ the smoothed Dirac delta function [2].

3 Segmentation and Pose Tracking

It is common in level-set region-based segmentation [2] to minimise an energy function of the form:

$$E(\Phi) = \int_{\Omega} H_e(\Phi) r_f(\mathbf{x}) + \big(1 - H_e(\Phi)\big) r_b(\mathbf{x}) d\Omega \tag{1}$$

where r_f and r_b are two monotonically decreasing functions measuring the matching quality of the image pixels with foreground and background models. The level-set embedding function $\Phi(\mathbf{x})$ is often taken to be a signed distance function (or approximation thereof), with positive values inside the foreground and negative values outside (i.e. in the background). Typically, r_f and r_b are given by the likelihood of a pixel property (such as colour) under a given model, i.e. $r(\mathbf{x}) = P(\mathbf{y}|M)$.

In contrast, [1] proposed a generative model of image formation that leads to a subtly different (and more effective) energy function which can be interpreted in terms of the posterior (as opposed to likelihood) of each pixel's foreground/background membership. Assuming pixel-wise independence, and replacing integration with summation, the energy given by the

negative log (posterior) probability of the shape of the contour (encoded by the embedding function Φ), given the image data, is:

$$\begin{aligned} P(\Phi|\Omega) &= \prod_{i=1}^{N} \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi) P_b\big) \Big) \Rightarrow \\ E(\Phi) &= -\log(P(\Phi|\Omega)) = -\sum_{i=1}^{N} \log \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi) P_b\big) \Big) \end{aligned} \tag{2}$$

where P_f and P_b are the posterior probabilities $P(M_f|\mathbf{y})$ and $P(M_b|\mathbf{y})$ respectively (further details are given in [1]).

In the normal level set formulation the contour would evolve in an unconstrained space so we would compute the derivative with respect to time of $E(\Phi)$. Instead, we differentiate with respect to the pose parameters λ_i, aiming to evolve the contour in a space parametrized by the pose parameters:

$$\frac{\partial E}{\partial \lambda_i} = P(\Phi|\Omega) \sum_{i=1}^{N} \frac{P_f - P_b}{H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b} \frac{\partial H_e(\Phi)}{\partial \lambda_i} \tag{3}$$

$$\frac{\partial H_e\big(\Phi(x,y)\big)}{\partial \lambda_i} = \frac{\partial H_e}{\partial \Phi} \left(\frac{\partial \Phi}{\partial x} \frac{\partial x}{\partial \lambda_i} + \frac{\partial \Phi}{\partial y} \frac{\partial y}{\partial \lambda_i} \right) = \delta_e(\Phi) \begin{bmatrix} \frac{\partial \Phi}{\partial x} & \frac{\partial \Phi}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \lambda_i} \\ \\ \frac{y}{\partial \lambda_i} \end{bmatrix} \tag{4}$$

At each iteration of the algorithm we re-compute Φ as the signed-distance transform from the projected contour (on the GPU for efficiency), and the differentials $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$ follow trivially.

Every 2D point $\mathbf{x}$ on the contour of the (true) projection of the 3D model has at least one corresponding 3D point $\mathbf{X}$:

$$\frac{\partial x}{\partial \lambda_i} = \frac{\partial}{\partial \lambda_i} \left(\frac{X}{Z} \right) = -\frac{1}{Z^2} \left(Z \frac{\partial X}{\partial \lambda_i} - X \frac{\partial Z}{\partial \lambda_i} \right) \tag{5}$$

and similarly for y. Noting that X, Y and Z are simple functions of the pose parameters, it is then straightforward, via a further application of the chain rule, to obtain $\frac{\partial X}{\partial \lambda_i}$, $\frac{\partial Y}{\partial \lambda_i}$ and $\frac{\partial Z}{\partial \lambda_i}$.

For image points on the inside of the contour, we backproject to the closest point on the object. For image points not exactly on the contour but outside (and which therefore have no true backprojection), we approximate $\mathbf{X}$ as the backprojection of the point on the contour which is closest in the image. Since the actual calculation takes place in a narrow band around the projected contour, this does not introduce gross errors.

When rotating and translating a 3D model some surfaces appear and others disappear, as the pose transitions between so-called "generic" views . The singular poses between generic views are characterised by ambiguous back-projections: a back-projected image contour point may have multiple tangencies with the object. In these cases the rate of change of the image contour with respect to the pose parameters is not only dictated by the closest points on the 3D model but also by the (infinitesimally invisible) furthest points (see Figure 2). In these transition cases we include terms from *both* the closest and distant points to give greater robustness when moving between different generic views of the object. To our knowledge

this aspect has not been considered in any of the previous literature on region-based 3D tracking.

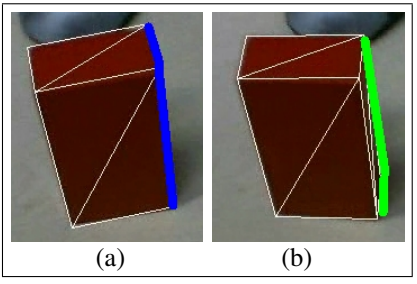


Figure 2: Importance of considering the most distant points. As the camera moves from (a) to (b) the evolution of the silhouette is not dictated by the blue points (closest points) but rather by the green points, which were the most distant points in (a).

A further difference with previous work is that we allow online adaptation of the foreground and background statistical models. Like [1], this is achieved using linear blending with variable learning rates α_i, $i = \{f, b\}$.

$$P_i(\mathbf{y}|M_i) \leftarrow (1-(\alpha_i))P_i(\mathbf{y}|M_i) + (\alpha_i)P_i(\mathbf{y}|M_i) \tag{6}$$

The factors α_i are fixed for all frames. In all our experiments $\alpha_f = 0.01$ and $\alpha_b = 0.02$. To avoid polluting the histograms with incorrect data, we only update when the energy minimization has converged.

4 Implementation and performance analysis

Here we discuss a number of implementational issues and analyse the computing requirements of the algorithm.

In the present work the appearance models are represented by RGB histograms using 32 bins per channel.

In practice we parametrize the rotation using quaternions. While this has the disadvantage of being an over-parametrisation, and with requirement to normalise the parameters periodically to 1 (we did this after each frame which took several iterations), it resulted in better performance than using the axis-angle representation.

We use simple gradient descent to minimize the energy function for expedience and ease of implementation. We hope to consider more sophisticated optimisation methods in the future. Each iteration of the gradient descent proceeds as follows:

- Render the 3D model with the current estimate of the pose. The rendering pipeline is similar to that of OpenGL, except that we generate *two* depth buffers: one for the closest points in the 3D model (from the camera) and one from the most distant points.
- Compute the contour of the projection and the exact signed distance transform of this contour, together with the two partial derivatives of the distance transform.
- Compute the partials of the image points with respect to the pose parameters.

Processing Stage	Required Time
Rendering the 3D model (not using the GPU)	2.4 ms
Transfer from host memory to GPU memory	0.5 ms
Finding the contour of the 2D projection	0.3 ms
Distance Transform of the contour of the 2D projection	1.6 ms
Computation of $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$	0.3 ms
Computation of all $\frac{\partial E}{\partial \lambda_i}$	0.6 ms

Table 1: Detailed proceeding time for one iteration of the minimization of our energy function

- Make a step change to the pose in the direction of the gradient. The step size is an adjustable parameter and was fixed for each model.

Most of the tasks involved in one iteration of our algorithm operate per-pixel and so we can achieve significant gains by exploiting this high degree of parallelism. We make use of the NVIDIA CUDA framework ([6]) to implement various steps on a GPU. Table 1 summarises the average amount of time taken by each stage of the evaluation, for a single iteration of the gradient descent. Because the processing is done by the GPU, the host CPU is free to do other tasks during processing. Therefore CPU speed only affects the 3D rendering step. In our experiments we used a Xeon E5420 CPU. Average processing time per iteration is around 5-8ms using a Geforce GTX 285 video card, for a 640x480 RGB image, depending on the size of the 3D rendering and the complexity of the 3D model.

We compute the contour of the projection by convolving the projection image with a Scharr kernel, and keeping only the pixels which are outside the projection. This implementation would have been impractical using a sequential algorithm, but it is very parallelisable. Our parallel implementation uses one thread per pixel.

For the signed distance transform we use an algorithm similar to that proposed by [4]. The distance transform is computed in two passes of the contour image: first by columns and then by rows. Our parallel implementation uses one thread for each column and then one thread for each row in the image. The second pass does not take full advantage of the CUDA framework capabilities because memory access is not coalesced. We are investigating using a different algorithm, which we hope will speed the distance transform by a factor of 2 or 3.

The computation of the partial derivatives of the signed distance transform uses central finite differences, parallelised by using one thread per pixel. Finally, the partial derivatives of the energy function are computed by using one thread per pixel and then transferred back to host memory.

The number of iterations needed for convergence depends on the complexity of the 3D model, on the the gradient descent step size and of course on the distance between the current pose estimate and the final pose estimate. The algorithm typically converges in up to 8 iterations per frame.

5 Results

We have tested our method on a variety of models and video sequences. In the following section we will demonstrate some of the qualities of our method, such as robustness to initialization and motion blur. We will also demonstrate the advantage of including the

most distant points of the object (which project to point the contour of the projection) in the evaluation of the energy function. Finally we will showcase the difference between using pixel-wise posteriors and the classical log likelihood formulation of level set. [1]

Figure 3 shows an example of our method running on a single frame [2]. In this example we are able to successfully recover scale changes of +/-40% and initial angular errors of 50 degrees on the x and y axis and 70 degrees on the z axis. Depending on the model and image data, in other tests we were able to recover rotations up to 90 degrees on each axis (depending on the model). With regard to translation, the gradient descent usually converges as long as the projection of the 3D model intersects the object in the image (also depending on the model and of course the image data itself).

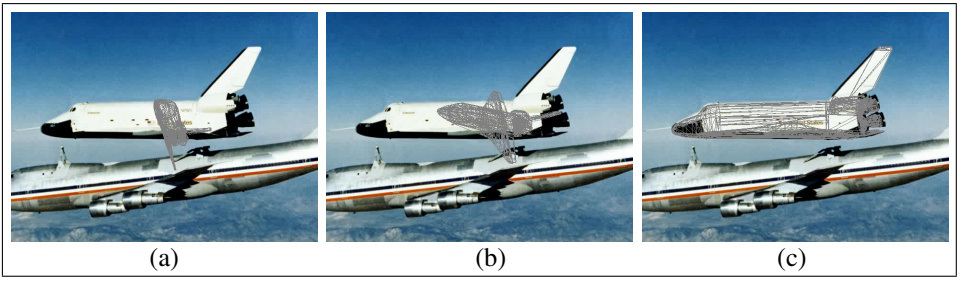


Figure 3: Typical algorithm run for one frame. (a) - initialization, (b) - gradient descent iteration, (c) - final result

One advantage of a region based tracker, when compared to an edge based tracker, is robustness to motion blur. In Figure 4 we use a soft-drink can and 3D model to showcase a typical scenario when our method is able to successfully track the 3D object while any local feature based pose recovery algorithm would most likely fail.

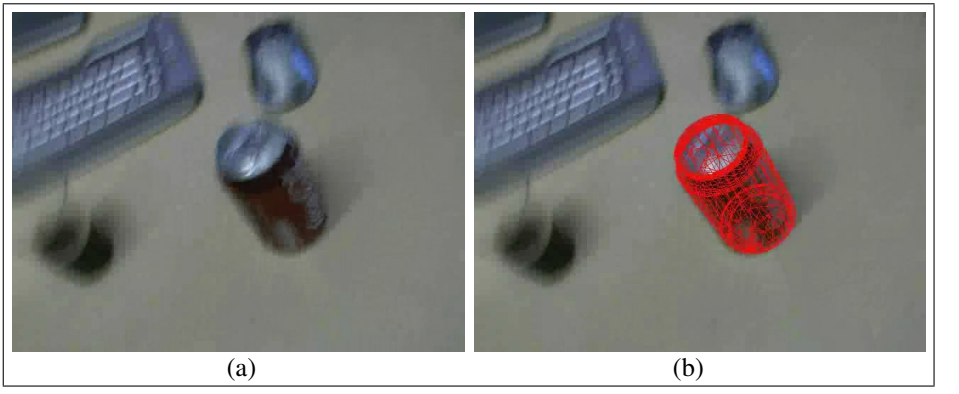
The evolution of the contour of the projection is not always dictated by the visible part of the object. Surfaces may appear and may disappear. To our knowledge there is no method in the literature on region based 3D tracking which models this behaviour. As mentioned in Section 3 our methods also considered the most distant points, which project to the the contour of the projection, in the energy function. In Figure 5 we show frames of a video sequence where the camera rotates above a soft-drink can. The view of the can moves to directly above, with the side obscured, back to a view with the side visible again. When including only the closest points as influencing the contour evolution, the pose incorrectly flips. In contrast the behaviour is correct when the furthest points are also considered.

A potential source of problems when using strong prior 3D (or 2D) shape knowledge in tracking and pose recovery, is discrepancies between the model and the actual target. In Figure 6 we show excerpts from a sequence in which we successfully track a human hand with an approximate model.

Rapid motions produce big differences in pose between successive frames. Because of the large basin of convergence our method can often recover the pose even in these cases. The algorithm does have slow convergence when surfaces appear and disappear so recovery of rapid motions is more difficult. Note that we do not use a motion model, though this could easily be incorporated and would greatly aid in the prediction of the appearance and

[1] We encourage the reader to view the supplementary video material in conjunction with this section.

[2] The 3D model of the space-shuttle was obtained from http://www.nasa.gov/.



(a) (b)

Figure 4: Robustness to motion blur

disappearance of surfaces.

We used our video sequences to perform a critical performance analysis where we looked at the characteristics of both the pixel-wise posteriors energy function ([1]) and the log likelihood one ([2]), for 3D tracking. For each frame we computed 120 perturbations around the correct pose (20 for each pose parameter) totaling more than 200000 experiments and we measured whether each algorithm converged and how many iterations were required for convergence to the true pose [3]. Figure 7 summarizes the results. Though there is little choice between the objective functions for recovery of translation in x and y axis (i.e. parallel to the image plane), the pixel-wise posteriors energy function outperforms the log likelihood objective function for recovering translation in the z axis and for all 3 rotation axis. Finally the number of iterations necessary for convergence (when convergence was possible) is considerably higher when using log likelihood (for both rotation and translation).

6 Conclusions

In this article we have proposed a novel level set based unified probabilistic framework for joint 2D segmentation and 2D–3D pose tracking. Our method allows for large variations in rotation (up to 90 degrees on each axis, depending on the 3D model) and translation to be successfully recovered. It has the typical qualities of region-based active contour techniques with shape priors, such as robustness to occlusion or noise.

In our ongoing research we are investigating a number of straightforward extensions to this work. In particular our method can also easily be extended to track and recover the pose of multiple objects from multiple views. Texture and edge information can also be easily incorporated into the 3D model and energy function. Finally the model can also be extended to segment, track and recover the poses of objects, given a stereo disparity map.

The additional information yielded by a 3D model-based tracker, versus a "dumb" 2D tracker such as [1] does come at the price of increased brittleness. A key issue facing our algorithm and all similar ones, is how to deal with the appearance of new surfaces or new object texture as the object rotates from one generic view to another. Such motions can radically change the appearance statistics of the foreground. Presently we can cope with this

[3] Without access to the ground truth, the true pose was measured subjectively.

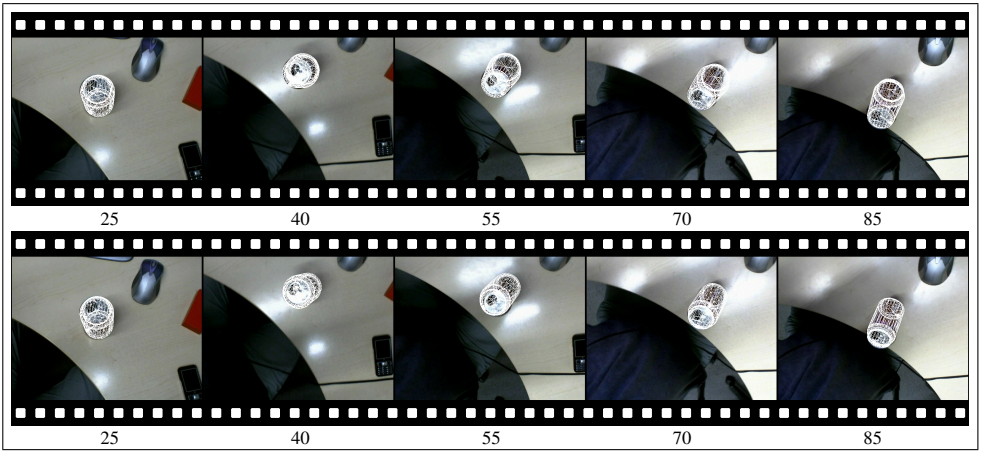


Figure 5: Film strip showing 5 frames from a video sequence of which the camera moves above the soft-drink can in a circular motion. Note that between frames 25 and 55 the view of the can moves to directly above, with the side obscured, back to a view with the side visible again. When including only the closest points as influencing the contour evolution, the pose incorrectly flips (above). In contrast the behaviour is correct when the furthest points are also considered (below).

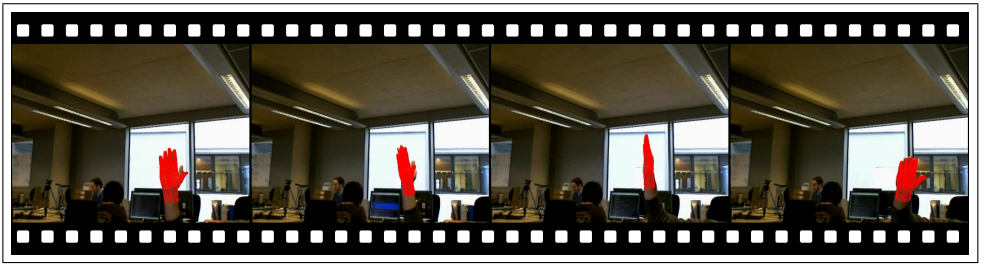


Figure 6: Film strip showing our method tracking a hand. Our algorithm can track the hand in this sequence at around 20fps.

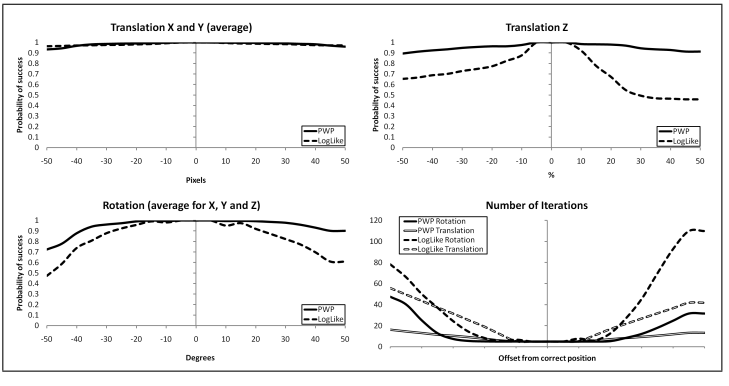


Figure 7: Performance analysis: Probability of convergence to the correct pose generated from over 200000 experiments

in many cases, because we explicitly take into account the expected contour evolution due to *all* points on the occluding contour (not just those which are the closest back-projection), and allow online appearance updating. However it seems likely that a simultaneous evolution in *appearance space* which predicts appearance changes (rather than simply adapting to them after the fact) would be beneficial. Further, a promising idea might be to combine 2D and 3D tracking, delivering 2D data and keeping lock on a target during times when full 3D pose recovery proves impossible.

References

[1] Charles Bibby and Ian Reid. Robust real-time visual tracking using pixel-wise posteriors. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 831–844, 2008.

[2] Daniel Cremers, Mikael Rousson, and Rachid Deriche. A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape. *International Journal of Computer Vision*, 72(2):195–215, 2007.

[3] Samuel Dambreville, Romeil Sandhu, Anthony Yezzi, and Allen Tannenbaum. Robust 3d pose estimation and efficient 2d region-based segmentation from a 3d shape prior. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 169–182, 2008.

[4] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, 2004.

[5] Chris Harris. Tracking with rigid models. *Active vision*, pages 59–73, 1993.

[6] NVIDIA. *NVIDIA CUDA Programming Guide 2.2*. 2009.

[7] Bodo Rosenhahn, Thomas Brox, and Joachim Weickert. Three-dimensional shape knowledge for joint image segmentation and pose tracking. *International Journal of Computer Vision*, 73(3):243–262, 2007.

[8] C. Schmaltz, B. Rosenhahn, T. Brox, D. Cremers, J. Weickert, L. Wietzke, and G. Sommer. Region-based pose tracking. In *Proc. 3rd Iberian Conference on Pattern Recognition and Image Analysis*, June 2007.

[9] Luminita A. Vese and Tony F. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International Journal of Computer Vision*, 50:271–293, 2002.