Int J Comput Vis (2012) 98:335–354
DOI 10.1007/s11263-011-0514-3

PWP3D: Real-Time Segmentation and Tracking of 3D Objects

Victor A. Prisacariu · Ian D. Reid

Received: 6 October 2010 / Accepted: 28 December 2011 / Published online: 10 January 2012
© Springer Science+Business Media, LLC 2012

Abstract We formulate a probabilistic framework for simultaneous region-based 2D segmentation and 2D to 3D pose tracking, using a known 3D model. Given such a model, we aim to maximise the discrimination between statistical foreground and background appearance models, via direct optimisation of the 3D pose parameters. The foreground region is delineated by the zero-level-set of a signed distance embedding function, and we define an energy over this region and its immediate background surroundings based on pixel-wise posterior membership probabilities (as opposed to likelihoods). We derive the differentials of this energy with respect to the pose parameters of the 3D object, meaning we can conduct a search for the correct pose using standard gradient-based non-linear minimisation techniques. We propose novel enhancements at the pixel level based on temporal consistency and improved online appearance model adaptation. Furthermore, straightforward extensions of our method lead to multi-camera and multi-object tracking as part of the same framework. The parallel nature of much of the processing in our algorithm means it is amenable to GPU acceleration, and we give details of our real-time implementation, which we use to generate experimental results on both real and artificial video sequences, with a number of 3D models. These experiments demonstrate the benefit of using pixel-wise posteriors rather than likelihoods, and showcase the qualities, such as robustness to occlusions and motion blur (and also some failure modes), of our tracker.

V.A. Prisacariu (✉) · I.D. Reid
Department of Engineering Science, University of Oxford, Oxford, UK
e-mail: victor@robots.ox.ac.uk

I.D. Reid
e-mail: ian@robots.ox.ac.uk

Keywords Level set · Region based · Pose recovery · 3D tracking · Cuda · GPGPU · Real time · Segmentation · Tracking

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, but there are few articles which study the tasks simultaneously and even fewer which consider the problem of real time performance. In the current work, we develop a method for model-based segmentation and tracking 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. We seek the six degree of freedom rigid transformation that maximises an energy function based on the posterior foreground and background membership probabilities of each pixel in the image, thus computing the pose and segmentation of the object simultaneously, using a single, unified energy function. We describe an implementation that achieves real-time performance.

From a segmentation point of view, our work can be seen as related to Cremers et al. (2006). Whereas Cremers et al. (2006) used a prelearned latent space of 2D shapes to constrain the evolution of the level set function, here we generate the shapes by projecting a 3D model. Like that work, our method is level set and region-based. However, unlike Cremers et al. (2006), and inspired instead by Bibby and Reid (2008), we aim to maximise the posterior per-pixel probability of foreground and background membership, as a function of pose. Bibby and Reid (2008) showed that this yields a better behaved energy function (in the 2D case), when compared to standard level set formulations such as Cremers et al. (2007). In our case we assume a known 3D model and

one or more calibrated cameras, and seek the six degree of freedom rigid transformation that maximises the pixel-wise posterior energy function.

Our primary interest lies in the pose tracking ability of the system. A comprehensive review of monocular model-based tracking was conducted by Lepetit and Fua (2005) and so here we consider only relevant highlights. A variety of features has been used for 3D pose tracking, including edges, point features, optic flow and regions. By far the most popular methods to date have been edge-based, in which the aim is to find the pose in which the projection of model edges aligns best with edges found in an image. This is perhaps because of the natural viewpoint invariance afforded by the edges, and also because the extent of edges in an image means that correspondence search can be done in 1D, rather than over two dimensions of the image as for point features. In this class of pose tracking systems the seminal work of Harris (1993) and Lowe (1992) inspired later similar efforts from Drummond and Cipolla (1999) and Marchand et al. (1999). These systems were all restricted to polyhedral and simple surface models. Point-based methods can model more general shapes and typically proceed by first establishing correspondences between projected point locations and image points. The rise of effective descriptors for point features that exhibit a degree of viewpoint invariance (e.g. Lowe 2004 or Lepetit et al. 2005) has seen real-time methods for point-based model-based tracking become more effective, such as Wagner et al. (2008) or Ozuysal et al. (2006) (where the model is also learned online), or even using the combination of points and edges, such as Rosten and Drummond (2005). Region-based methods such as Hager and Belhumeur (1998), or Jurie and Dhome (2002) are more accurately categorised as 2D–2D template trackers, since 6D pose of a planar object is estimated, rather than a fully 3D object. In contrast, region-based methods such as Rosenhahn et al. (2007) seek alignment of the projection of the occluding boundaries of the object with a regional segmentation of the image. This approach has the advantage that it tends to be less disrupted by occlusions, clutter and motion blur than edge-based methods. Furthermore, these techniques have been shown to work with arbitrarily shaped objects. One issue when using only regions, however, is that often a good segmentation is required before the pose recovery, meaning that a poor segmentation will lead to poor pose recovery results. Also, the silhouette to pose mapping is multimodal i.e. the same silhouette can be generated by multiple poses.

The most closely related work to our own is based on simultaneous segmentation and tracking using region-based criteria (Rosenhahn et al. 2007; Schmaltz et al. 2007a; Dambreville et al. 2008). Rosenhahn et al. (2007) adapt an infinite dimensional active contour in a single iterative step, in two stages: first the contour, represented by a zero level-set of a 2D embedding function with a shape term (to encourage similitude between the segmentation and the expected silhouette of the object given the current pose), 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 is sought that best satisfies the tangency constraints that exist between the 3D object and these rays. In some respects it bears similarity to Harris' RaPiD system (Harris 1993), with the level set evolution to find the extremal contour replacing Harris' 1D searches for edges. This two-stage iteration places only soft constraints on the evolution of the contour i.e. the minimisation of the energy function is done in an infinite dimensional space (with some encouragement), rather than in the space of possible contours. Furthermore our experience with the two-stage iteration is that the 2D–3D pose matching does not have a large basin of convergence. This method is extended in Brox et al. (2009) where the region statistics are augmented with SIFT features and optical flow and in Gall et al. (2008) where an image synthesis stage is added.

Schmaltz et al. (2007a) proposed a cleaner approach. 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 and background membership likelihoods of each point, while the amount of evolution is apparently a "tunable" parameter. The pose is then determined using the same method as Rosenhahn et al. (2007).

The work most closely related to ours, Dambreville et al. (2008), 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 by Vese and Chan (2002).

In our work we use the pixel-wise posterior foreground and background membership approach of Bibby and Reid (2008) (which would not yield a convenient collapse of the above-mentioned term). Rather than formulate our energy over the sum of separate integrals for foreground and background, we use the Heaviside step function, evaluated on the 2D 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 while still allowing for real-time speeds; and (ii) more importantly, it is

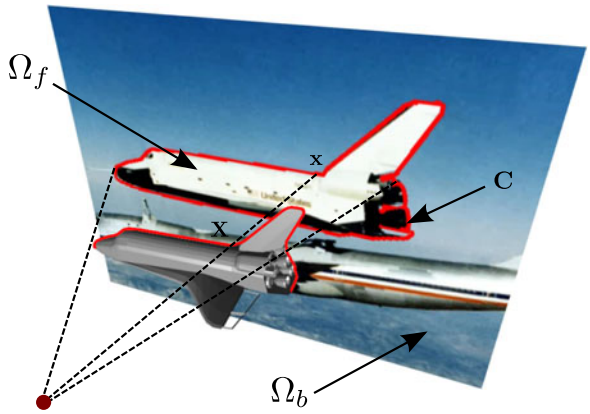

Fig. 1 Representation of the object showing: the contour of the projection **C** and the corresponding contour around the visible part of the 3D model, the foreground region Ω_f and the background region Ω_b, a 2D point **x** on the contour and its corresponding 3D point in the object coordinate frame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

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.

An early version of this work was presented in a preliminary conference paper, Prisacariu and Reid (2009). In the present paper we elaborate this work, show extensions to multiple objects and views, compare multiple minimisation algorithms, add temporal consistency to the per pixel posteriors probabilities and provide more detailed experiments.

The remainder of the paper is structured as follows: we begin by briefly describing the geometry and setting up our notational conventions, in Sect. 2. In Sect. 3 we provide a basic introduction to level set functions. In Sect. 4 we describe the mathematical foundations, while in Sect. 5 we derive our algorithm and discuss its implementation. In Sect. 6 we consider various qualitative and quantitative evaluations of our method, on both real and synthetic data, and we conclude in Sect. 7.

2 Notation

Let the image be denoted by $\mathbf{I}$ and the image domain by $\Omega \subset \mathbb{R}^2$. An image pixel $\mathbf{x} = [x, y]$ has a corresponding image value $\mathbf{I}(\mathbf{x}) = \mathbf{y}$ (in our experiments an RGB value), a 3D point $\mathbf{X} = [X, Y, Z]^T = \mathbf{R}\mathbf{X}_0 + \mathbf{T} \in \mathbb{R}^3$ in the camera coordinate frame and a point $\mathbf{X}_0 = [X_0, Y_0, Z_0]^T \in \mathbb{R}^3$ in the object coordinate frame. $\mathbf{R}$ and $\mathbf{T}$ are the rotation matrix and translation vector respectively (representing the unknown pose) and are parameterised by 7 parameters (4 for rotation and 3 for translation), denoted by λ_i. We use quaternions to represent rotation, so we use 4 parameters to capture the 3 degrees of freedom of the rotation.

We assume the camera calibration parameters (for each camera) to be known. Let (f_u, f_v) be the focal distance expressed in horizontal and vertical pixels (Bouguet 2008) and (u_o, v_o) the principal point of the camera. In the case of multiple cameras, the extrinsics were obtained using the Matlab Calibration Toolbox (Bouguet 2008).

The contour around the visible part of the object in 3D projects to the contour **C** in the image (marked with red in Fig. 1). We embed **C** as the zero level-set of the function $\Phi(\mathbf{x})$ (Osher and Sethian 1988). The contour **C** also segments the image into two disjoint regions: foreground denoted by Ω_f and background denoted by Ω_b. Each region has its own statistical appearance model, $P(\mathbf{y}|M_f)$ for foreground and $P(\mathbf{y}|M_b)$ for background.

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

3 Level Set Functions

Level set functions (Osher and Sethian 1988) provide a simple framework for modelling the shape and evolution of curves. Curves break up, merge, move or disappear during the course of their evolution. When using the conventional representation i.e. *explicit* representation of a curve, complicated methods have to be developed to model its behaviour. The level set method handles all these topological changes very easily.

Consider a 2D closed curve **C**, separating a domain Ω into an interior region Ω_f and an exterior region Ω_b (like the ones referred to in Sect. 2). A level set function Φ is a Lipschitz continuous function, implicitly defining a curve as its zero level. For our 2D closed curve **C**, this is to say that $\mathbf{C} = \{(x, y) \in \mathbb{R}^2 | \Phi(x, y) = 0\}$ corresponds to the location of the embedded curve. This is called the *implicit* representation of the curve **C**.

In remainder of this work we use a subset of these implicit functions, namely *signed distance functions*. A distance function $d(\mathbf{x})$ is defined as:

$$d(\mathbf{x}) = \min_{\mathbf{x}_c \in \mathbf{C}} |\mathbf{x} - \mathbf{x}_c| \tag{1}$$

A signed distance function is a level set function for which $\Phi(\mathbf{x}) = -d(\mathbf{x}), \forall \mathbf{x} \in \Omega_f$ and $\Phi(\mathbf{x}) = d(\mathbf{x}), \forall \mathbf{x} \in \Omega_b$. These impose an extra condition on the level set function, namely $|\nabla \Phi(x, y)| = 1$.

Figure 2 shows an example of a level set embedding function defined on a 2D domain.

4 Segmentation and Pose Tracking

It is common in region-based segmentation for a closed curve to be evolved such that the discrepancy between the

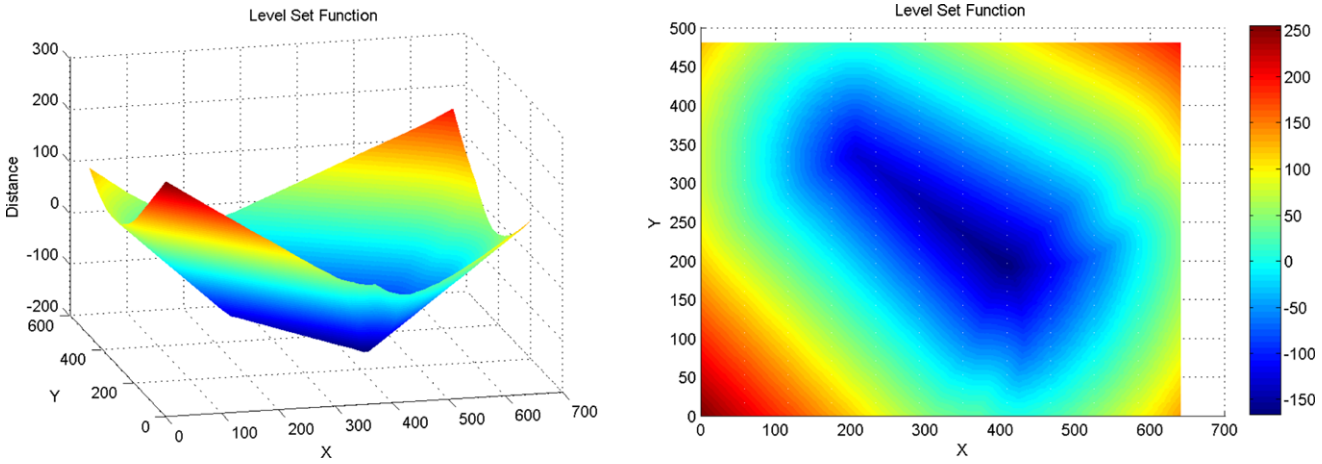

Fig. 2 Example level set function embedding the contour of the projection of a soft-drink can 3D model. The size of the image containing the contour is 640×480

statistics of the foreground region and those of the background region is maximised:

$$E = \int_{\Omega_f} r_f(I(\mathbf{x}), \mathbf{C}) d\Omega + \int_{\Omega_b} r_b(I(\mathbf{x}), \mathbf{C}) d\Omega \tag{2}$$

where r_f and r_b are two monotonically decreasing functions, measuring the matching quality of the image pixels with respect to the foreground and background models.

In level-set region-based segmentation (Cremers et al. 2007) the energy function is rewritten as:

$$E(\Phi) = \int_{\Omega} H_e(\Phi) r_f(\mathbf{x}) + \big(1 - H_e(\Phi)\big) r_b(\mathbf{x}) d\Omega \tag{3}$$

As stated before, the level-set embedding function $\Phi(\mathbf{x})$ is taken to be a signed distance function (or an approximation thereof).

Typically, r_f and r_b are given by the likelihood of a pixel property (such as colour) under a given model, i.e. $r(\mathbf{x}) = P(\mathbf{y}|M)$, with M being either M_f or M_b. In contrast, Bibby and Reid (2008) 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 or 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:

$$P(\Phi|\mathbf{I}) = \prod_{\mathbf{x}\in\Omega} \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b \Big) \quad \Rightarrow \tag{4}$$

$$E(\Phi) = -\log(P(\Phi|\mathbf{I})) \tag{5}$$

$$= -\sum_{\mathbf{x}\in\Omega} \log \Big(H_e(\Phi) P_f + \big(1 - H_e(\Phi)\big) P_b \Big) \tag{6}$$

where P_f and P_b are defined as:

$$P_f = \frac{P(\mathbf{y}|M_f)}{\eta_f P(\mathbf{y}|M_f) + \eta_b P(\mathbf{y}|M_b)} \tag{7}$$

$$P_b = \frac{P(\mathbf{y}|M_b)}{\eta_f P(\mathbf{y}|M_f) + \eta_b P(\mathbf{y}|M_b)} \tag{8}$$

with η_f and η_b being the areas of the foreground and background regions respectively:

$$\eta_f = \sum_{\mathbf{x}\in\Omega} H_e\big(\Phi(\mathbf{x})\big) \qquad \eta_b = \sum_{\mathbf{x}\in\Omega} \Big(1 - H_e\big(\Phi(\mathbf{x})\big)\Big) \tag{9}$$

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

$$\frac{\partial E}{\partial \lambda_i} = -\sum_{\mathbf{x}\in\Omega} \frac{P_f - P_b}{H_e(\Phi) P_f + (1 - H_e(\Phi)) P_b} \frac{\partial H_e(\Phi)}{\partial \lambda_i} \tag{10}$$

$$\frac{\partial H_e(\Phi(x,y))}{\partial \lambda_i} = \frac{\partial H_e}{\partial \Phi} \left(\frac{\partial \Phi}{\partial x} \frac{\partial x}{\partial \lambda_i} + \frac{\partial \Phi}{\partial y} \frac{\partial y}{\partial \lambda_i} \right) \tag{11}$$

$$= \delta_e(\Phi) \begin{bmatrix} \frac{\partial \Phi}{\partial x} & \frac{\partial \Phi}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \lambda_i} \\ \frac{\partial y}{\partial \lambda_i} \end{bmatrix} \tag{12}$$

Note that P_f and P_b depend on the ratio between the η_f and η_b areas, so they potentially change with the shape Φ and the pose λ, in particular with the pose parameter λ governing scale. However, the derivatives of P_f and P_b are numerically very small. For example, the total value of any $\frac{\partial P_f}{\partial \lambda_i}$ is smaller than that of a single per point derivative of the rest of the energy function. We therefore treat them as constants.

At each iteration of the algorithm we re-compute Φ as the signed-distance function from the projected contour. The differentials $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$ follow trivially, using centred finite differences.

Every 2D point $\mathbf{x}$ on the contour of the (true) projection of the 3D model has at least one corresponding 3D point $\mathbf{X}$. For each of these points we can write:

$$\begin{bmatrix} x \\ y \end{bmatrix}^T = \begin{bmatrix} f_u \frac{X}{Z} + u_0 \\ f_v \frac{Y}{Z} + v_0 \end{bmatrix}^T \tag{13}$$

Therefore:

$$\begin{aligned} \frac{\partial x}{\partial \lambda_i} &= f_u \frac{\partial}{\partial \lambda_i} \frac{X}{Z} = f_u \frac{1}{Z^2} \left(Z \frac{\partial X}{\partial \lambda_i} - X \frac{\partial Z}{\partial \lambda_i} \right) \\ \frac{\partial y}{\partial \lambda_i} &= f_v \frac{\partial}{\partial \lambda_i} \frac{Y}{Z} = f_v \frac{1}{Z^2} \left(Z \frac{\partial Y}{\partial \lambda_i} - Y \frac{\partial Z}{\partial \lambda_i} \right) \end{aligned} \tag{14}$$

We choose to parameterise the rotation with quaternions. While this has the disadvantage of being an overparameterisation, and with requirement to normalise the parameters periodically to 1 (we do this after each frame), it resulted in better performance over using the axis-angle representation.

Every 3D point $\mathbf{X}$ in camera coordinates has a corresponding 3D point $\mathbf{X}_0$ in object coordinates:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \\ 0 \end{bmatrix} \tag{15}$$

with

$$\mathbf{R} = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_x q_y - 2q_z q_w & 2q_x q_z + 2q_y q_w & 0 \\ 2q_x q_y + 2q_z q_w & 1 - 2q_x^2 - 2q_z^2 & 2q_y q_z - 2q_x q_w & 0 \\ 2q_x q_z - 2q_y q_w & 2q_y q_z + 2q_x q_w & 1 - 2q_x^2 - 2q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

where q_x, q_y, q_z, q_w are the quaternion rotation parameters and t_x, t_y, t_z are translation parameters.

Finally, the differentials of the 3D points in camera space with respect to the pose parameters are shown in Table 1.

4.1 Optimisation Algorithm

There are several optimisation methods at our disposal. In this work we detail our results with gradient descent and conjugate gradients.

The obvious first choice is gradient descent. This method uses only the first derivative, yielding fast iterations. Unfortunately a very small step size must be used to avoid jumping over minima. We use a step size of approximately 0.5 degrees for rotation and 10 pixels in either direction (including z scaled with focal length), for translation. These step sizes were chosen experimentally.

The nonlinear conjugate gradients method uses only the first derivative (like gradient descent) but instead of going downhill in the direction of the local gradient, it uses the conjugate direction. To obtain the conjugate of the current direction, a β function is defined and the direction d_k at step k is computed as $d_k = -g_k + d_{k-1}\beta_k$, where g_k is the value of the derivative of the function to be minimised and d_{k-1} is the direction used at the previous step. The two most popular choices (Hager and Zhang 2005) for this function are the Polack-Ribiere $\beta_k^{PR} = \frac{\langle g_k, g_k - g_{k-1} \rangle}{\|g_{k-1}\|^2}$ and the Fletcher-Reeves $\beta_k^{FR} = \frac{\|g_k\|^2}{\|g_{k-1}\|^2}$ formulae, where $\langle \cdot, \cdot \rangle$ is the scalar product, $\| \cdot \|^2$ is the L_2 norm and g_k and g_{k-1} are the derivatives of the function to be minimised at the current and previous iterations. Throughout our testing we noted that the Polack-Ribiere formula leads to faster convergence in the case of rotation, but the Fletcher-Reeves function often leads to faster convergence in the case of translation. We did not observe the Fletcher-Reeves jamming phenomenon (Hager and Zhang 2005), but we did have cases where the Polack-Ribiere formula did not lead to a correct minimum. In this work we use the hybrid β function proposed by Gilbert and Nocedal (1992) which, throughout our testing, provided the best results of both β_k^{FR} and β_k^{PR} while still yielding convergence: $\beta_k^{GN} = \max\{-\beta_k^{FR}, \min\{\beta_k^{PR}, \beta_k^{FR}\}\}$.

We do not use any preconditioning. The standard preconditioner used in nonlinear conjugate gradients is the Hessian matrix, or an approximation obtained by a Broyden-Fletcher-Goldfarb-Shanno (BFGS) iteration (Press 2007). Since a preconditioner needs to be positive definite, we need to test if our Hessian matrix is positive definite at every iteration. This, along with the computation of the Hessian matrix, increases iteration time considerably, without leading to a significant reduction in the number of iterations required for convergence.

Another problem with nonlinear conjugate gradients is that if the function is not close to quadratic, iterations tend to lose conjugacy and require a reset in the steepest gradient direction. This is usually done after a preset number of iterations. In this work we evaluate (at every iteration) the energy function after a gradient descent iteration and after a conjugate gradient descent iteration and choose the one which leads to the smallest value for the energy function. This guarantees fast convergence but increases the computation time for each iteration, as the energy function needs to be evaluated twice at each iteration.

The number of iterations needed for convergence depends on the algorithm used, on the complexity of the 3D model, on the step size and, of course, on the distance between the current pose estimate and the final pose estimate. For a simple model (with few triangles), like the one shown

Table 1 Partial differentials of the 3D point in the camera frame **X** with respect to the pose parameters

λ_i	$\frac{\partial X}{\partial \lambda_i}$	$\frac{\partial Y}{\partial \lambda_i}$	$\frac{\partial Z}{\partial \lambda_i}$
t_x	1	0	0
t_y	0	1	0
t_z	0	0	1
q_x	$2q_yY_0 + 2q_zZ_0$	$2q_yX_0 - 4q_xY_0 - 2q_wZ_0$	$2q_zX_0 + 2q_wY_0 - 4q_xZ_0$
q_y	$2q_xY_0 - 4q_yX_0 + 2q_wZ_0$	$2q_xX_0 + 2q_zZ_0$	$2q_zY_0 - 2q_wX_0 - 4q_yZ_0$
q_z	$2q_xZ_0 - 2q_wY_0 - 4q_zX_0$	$2q_wX_0 - 4q_xY_0 + 2q_yZ_0$	$2q_xX_0 + 2q_yY_0$
q_w	$2q_yZ_0 - 2q_zY_0$	$2q_zX_0 - 2q_xZ_0$	$2q_xY_0 - 2q_yX_0$

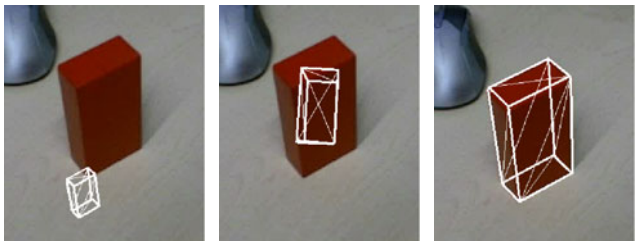

Fig. 3 Setup (image, model used and initialisations) for the convergence experiment

in Fig. 3, when using gradient descent, the algorithm typically converges in up to 8 iterations per frame (in the case of a fixed step) or 2–3 when a line search is used. More complex models require more iterations. For example, for a 3D model of a hand, like the one in Fig. 5, the algorithm typically converges within 20 simple gradient descent iterations. Since our algorithm is region-based, the convergence speed is dictated by the size of the regions. If the pose is defined by a small feature on the silhouette, e.g. the tip of a finger from a hand, convergence will be slower, because the area of overlap will be smaller.

The speed of convergence also depends on how close the energy function is to a quadratic function. When recovering translation (i.e. when the energy function is close to being quadratic) we observed that conjugate gradients achieves convergence about 3 times faster than gradient descent. In the rotation case (when the energy function is far from being quadratic) the conjugate gradients needs more resets and actual convergence is slower (in time) than gradient descent, even though the number of iterations is still smaller. In Table 2 we depict our timing results for the translation and rotation cases on a sample case where both rotation and translation were very far from the correct values (50 degrees on the x and y axis and 70 degrees on the z axis for rotation and about 5% overlap for translation). Note that this is an extreme case and for tracking the starting point would typically be an order of magnitude or more closer to the true value, with corresponding less computational effort (see Sect. 6). The setup for this experiment is depicted in Fig. 3.

To summarise, while conjugate gradients does often lead to a smaller number of iterations (as high as 3 times less iterations than gradient descent), it also takes 2–3 times the processing time of gradient descent, per iteration. In our experiments we use gradient descent unless otherwise stated.

To check for convergence we consider the magnitude of the derivatives of the energy function with respect to the pose parameters. When it is smaller than a preset threshold (15 in our tests) we stop iterating.

4.2 Multiple Objects

To track multiple objects we minimise a separate energy function (6) for each object. Each object has its own level set embedding function Φ and foreground and background membership probability (therefore also its own normalising factors η_f and η_b and foreground and background histograms, $P(\mathbf{y}|M_f)$ and $P(\mathbf{y}|M_b)$). The object-level energy function becomes:

$$E^j(\Phi^j) = -\sum_{\mathbf{x}\in\Omega} \log\left(H_e(\Phi^j)P_f^j + \left(1 - H_e(\Phi^j)\right)P_b^j\right) \quad (17)$$

where j is the object.

When tracking multiple objects quite often they occlude each other, or at least parts of each other. If pixels belonging to an object are considered in the energy function of another object, an incorrect pose will be obtained for the second object. Since we extract the full 6 DoF pose of each object, reasoning about the relative depths and occlusions in the multi-object case is trivial. We use the current estimate of the poses to render all tracked objects and select only the visible pixels from each 2D projection, in a manner similar to Schmaltz et al. (2007b).

4.3 Multiple Views

Our system can use multiple views of the same 3D object(s) to achieve better tracking. All the pixels, from all the views, contribute towards the minimisation of a *single* energy function (for each object), yielding a *single* set of pose parameters. There are however different foreground and background probabilities and different level sets functions. For

Table 2 Number of iterations and timing details required for convergence when using gradient descent with (GD) and without (GD-LS) a line search step and conjugate gradients with (CG) and without (CG-LS) a line search step. The time per iteration for translation is smaller than that for rotation because computing the derivatives with respect to rotation requires considerably more mathematical operations (about 10 times more). The setup for this experiment is depicted in Fig. 3

	Translation				Rotation			
	GD	GD-LS	CG	CG-LS	GD	GD-LS	CG	CG-LS
Number of iterations	120	40	39	36	132	60	57	54
Processing time per iteration (ms)	4.45	12.2	13.5	14.1	4.65	12.6	13.8	14.2
Total time for convergence (ms)	534	489	528	508	613	760	791	769

V views, the energy function becomes:

$$E(\Phi^1,\ldots,\Phi^V) = -\sum_{v=1}^{V}\sum_{\mathbf{x}\in\Omega}\log\left(H_e(\Phi^v)P_f^v + \left(1-H_e(\Phi^v)\right)P_b^v\right) \quad (18)$$

where v is the view.

When tracking multiple objects from multiple views, the object-level energy function becomes:

$$E^j(\Phi^{j,1},\ldots,\Phi^{j,V}) = -\sum_{v=1}^{V}\sum_{\mathbf{x}\in\Omega}\log\left(H_e(\Phi^{j,v})P_f^{j,v} + \left(1-H_e(\Phi^{j,v})\right)P_b^{j,v}\right) \quad (19)$$

where j is the object and v is the view.

We obtained the intrinsic camera parameters for each camera and the extrinsic parameters between the cameras using the Matlab Calibration Toolbox (Bouguet 2008).

4.4 Invisible Points

When a 3D model is rotated or translated, surfaces may appear or disappear, as the pose transitions between so-called "generic" views (Freeman 1993; Binford 1981). 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 (from the camera) but also by the (infinitesimally invisible) furthest points (see Fig. 4). In these transition cases we include terms from *both* the closest and the most 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.

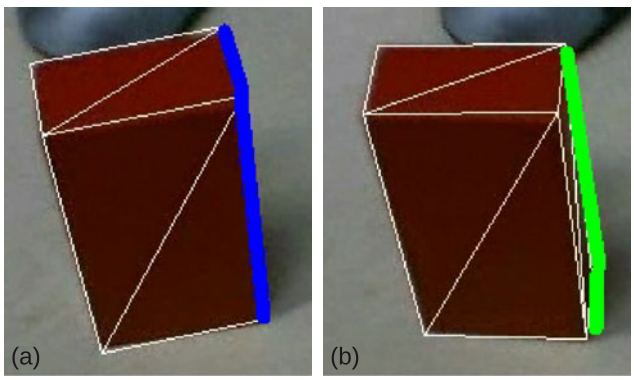

Fig. 4 Importance of considering the most distant points. As the camera moves from (**a**) to (**b**) the evolution of the silhouette is not dictated by the closest points, highlighted in (**a**), but rather by the most distant point, highlighted in (**b**)

Equation (14) becomes:

$$\begin{aligned}\frac{\partial x}{\partial\lambda_i} &= f_u\frac{1}{Z_c^2}\left(Z_c\frac{\partial X}{\partial\lambda_i} - X\frac{\partial Z_c}{\partial\lambda_i}\right) + f_u\frac{1}{Z_d^2}\left(Z_d\frac{\partial X}{\partial\lambda_i} - X\frac{\partial Z_d}{\partial\lambda_i}\right)\\ \frac{\partial y}{\partial\lambda_i} &= f_v\frac{1}{Z_c^2}\left(Z_c\frac{\partial Y}{\partial\lambda_i} - Y\frac{\partial Z_c}{\partial\lambda_i}\right) + f_v\frac{1}{Z_d^2}\left(Z_d\frac{\partial Y}{\partial\lambda_i} - Y\frac{\partial Z_d}{\partial\lambda_i}\right)\end{aligned} \quad (20)$$

where Z_c and Z_d are the depth values for the closest point and most distant points in the model that project to $[x, y]$.

While we do acknowledge that only either the closest or the most distant point (not both of them) may influence the contour at any given time, checking all possible hypotheses would be intractable, because for each pixel in the 3D rendering both hypotheses would need to be checked, leading to a total of 2^N hypotheses, where N is the total number of pixels inside the 3D rendering. Our solution provides a good approximation, as proven empirically in Sect. 6.

4.5 Temporal Consistency

So far we have treated pixels as being independent in space and time. The former could be dealt with using a MRF, but at the cost of much slower evaluation. In contrast here we introduce temporal dependence using a recursive Bayes filter:

$$P(M_j^t|\mathbf{Y}^t) = \frac{P(\mathbf{y}^t|M_j^t, \mathbf{Y}^{t-1})P(M_j^t|\mathbf{Y}^{t-1})}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \tag{21}$$

where M_j, $j \in \{f, b\}$ is the foreground/background model, $\mathbf{y}^t$ the value of pixel $\mathbf{y}$ at time t and $\mathbf{Y}_t = [\mathbf{y}^t, \mathbf{y}^{t-1}, \ldots]$ the values of pixel $\mathbf{y}$ up to time t.

Following the Bibby and Reid (2008) generative model, we assume conditional independence and write:

$$\begin{aligned} &P(M_j^t|\mathbf{Y}^t) \\ &= \frac{P(\mathbf{y}^t|M_j^t)P(M_j^t|\mathbf{Y}^{t-1})}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \\ &= \frac{P(\mathbf{y}^t|M_j^t)\sum_{i\in\{f,b\}}(p(M_j^t|M_i^{t-1})p(M_i^{t-1}|\mathbf{Y}^{t-1}))}{P(\mathbf{y}^t|\mathbf{Y}^{t-1})} \end{aligned} \tag{22}$$

where

$$P(\mathbf{y}^t|\mathbf{Y}^{t-1}) = \sum_{i\in\{f,b\}} P(\mathbf{y}^t|M_i^t)P(M_i^t|\mathbf{Y}^{t-1}) \tag{23}$$

The recursion is initialised by $P(M_0) = P(M_j)$.

To estimate $P(M_j^t|M_i^{t-1})$ for $i \in \{f, b\}$ and $j \in \{f, b\}$ we write:

$$P(M_j^t|M_i^{t-1}) = \frac{\zeta_{ji}}{\zeta} \tag{24}$$

where ζ is the total number of pixels in the image and ζ_{ji}, $i \in \{f, b\}$ and $j \in \{f, b\}$ is the number of pixels that were of type i at the previous frame and are of type j at the current frame. For example a pixel is considered foreground (i.e. of type f) at time t if $P(\mathbf{y}^t|M_f^t) > P(\mathbf{y}^t|M_b^t)$.

In this work we only enforce first order temporal consistency i.e. $\mathbf{Y}^t = \mathbf{y}^t$, which means that the current value of the foreground or background posterior depends only on the value at the previous time step.

4.6 Online Adaptation

In this work we use the same adaptation equation as our previous work, Prisacariu and Reid (2009).

$$P^t(\mathbf{y}|M_i) = (1-\alpha_i)P^{t-1}(\mathbf{y}|M_i) + \alpha_i P^t(\mathbf{y}|M_i) \tag{25}$$

where $i = \{f, b\}$.

In our earlier work (Prisacariu and Reid 2009), we updated the histogram only when the minimisation of the energy function had converged. This does not guarantee that the histograms will not be corrupted, as the minimisation of the energy function might have converged to an incorrect pose. Here we use the distance transform as a measure of the uncertainty in the contour: pixels far from the contour and inside the projection will most likely be foreground, while pixels far from the contour and outside the projection will most likely be background. We set a threshold distance and update the foreground and background histograms only where the absolute value of the distance transform is bigger than this threshold. This threshold is equal to the size of the band in which we evaluate energy function (fixed at 16 pixels, in all our experiments). While this might mean that we do not update with some pixels that actually are foreground or background, it does allow us to use much higher adaptation rates and ensure more stable tracking. In Prisacariu and Reid (2009) the foreground adaptation rate was usually less than 1% ($\alpha_f = 0.01$). Here we can easily increase the foreground learning rate to as high as 5% ($\alpha_f = 0.05$) and tracking is still stable (the histograms do not get corrupted). This allows more rapid changes in appearance which in turn produce more reliable tracking. The background adaptation rate remains unchanged ($\alpha_b = 0.02$), because the corrupting influence of the foreground pixels was already negligible.

5 Implementation and Performance Analysis

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

In the present work the appearance models are represented by RGB histograms using 32 bins per channel. We use 640 × 480 images for all experiments. Each iteration 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. To generate the most distant points we use a reversed Z-buffer algorithm i.e. instead of choosing only the pixel with the smallest depth, we choose the one with the biggest depth.
- Compute the contour of the projection and the signed distance transform of this contour.
- Compute the partial derivatives of the signed distance transform.
- Compute the partial derivatives of the energy function.
- In the gradient descent case: make a step change to the pose in the direction of the gradient. The step size may be found using a line search algorithm or be a preset adjustable parameter.

Fig. 5 The average processing time when using the model on top with the image on the bottom is 2.91 ms. The resolution of the images is 640 × 480 and the model has 680 *triangles*

– In the conjugate gradient case: make a step change to the pose in the direction of the gradient and in the direction of the conjugate gradient. Evaluate the energy function in both cases and choose the minimum. The step size may also be found using a line search algorithm or be a preset adjustable parameter.

Most of the tasks involved in one iteration of our algorithm operate per-pixel, so we can achieve significant gains by exploiting the high degree of parallelism. We use the CUDA framework (NVIDIA 2009) to implement various steps on a GPU. Average processing time per iteration (when using gradient descent) lies between 2.5 and 5 ms using a Geforce GTX 285 video card, for a 640 × 480 RGB image, depending on the size of the 3D rendering and the complexity of the 3D model. The more pixels being rendered, the bigger the processing time. However, the energy function is only evaluated in a band around the contour so the algorithm scales well with image size. For example, for the model shown in Fig. 5, the average processing time per iteration is 2.91 ms. Table 3 summarises the average amount of time taken by each stage of the evaluation, for a single iteration of the gradient descent. Since the processing is done by the GPU, the host CPU is free to do other tasks during processing.

Table 3 Detailed processing time for one iteration of the minimisation of our energy function, when using gradient descent without the line search (so no evaluation of the energy function is necessary)

Processing stage	Time
Rendering the 3D model	0.64 ms
Finding the contour of the 2D projection	0.18 ms
Distance transform of the contour of the 2D projection	1.50 ms
Computation of $\frac{\partial \Phi}{\partial x}$ and $\frac{\partial \Phi}{\partial y}$	0.15 ms
Computation of all $\frac{\partial E}{\partial \lambda_i}$	0.44 ms

5.1 3D Rendering Engine

For every iteration of our algorithm, the 3D model must be rendered with a given pose and projection matrix. The distance transform needs to be computed only in the region of the image where the object is located. If a standard rendering engine is used, this region of interest must be computed after the rendering. Also, to our knowledge, it is not straightforward for OpenGL or DirectX to render multiple Z-buffers (requiring multiple separate renderings). We chose to implement our own 3D rendering engine so the region of interest is computed at *render time*. We can also easily add support for multiple Z-buffers. Finally, since there is no need to consider lighting or texturing (which would have been costly in a CPU implementation), our custom rendering engine operates at a similar speed to OpenGL or DirectX (a difference of less than 1 ms, in our tests).

Figure 6 shows the steps undertaken by our rendering engine to draw a single image pixel, from a known 3D point. Just like OpenGL, first the 3D point must be converted from the object coordinate system to the camera coordinate system, using (15). In the second and third steps (13) is used to obtain the *normalised device coordinates* for the x and y coordinates of the pixels. The normalised z coordinates have the following expression:

$$z_{NDC} = \frac{f+n}{f-n} + \frac{2fn}{f-n}\frac{1}{Z} \tag{26}$$

where f and n are the distances with respect to the far and near planes, respectively. We use $f = 400$ and $n = 1$. In this coordinate system the x, y and coordinates are mapped to the range $(-1, 1)$. This is a non-linear mapping, meaning that closer points have higher resolution than more distant points.

In the viewpoint transform the 3D coordinates are mapped into the 2D area of the rendering window.

We assume the 3D model to be a triangle mesh. We use interpolation to obtain the x, y and z values for the points

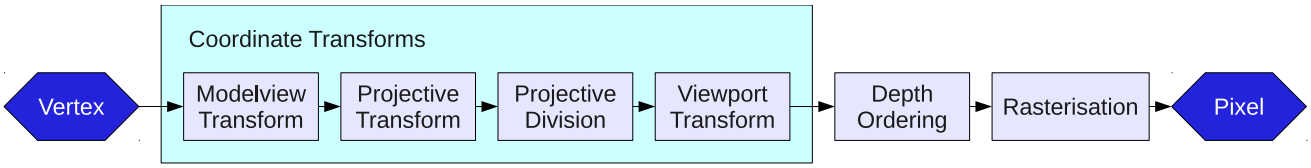

Fig. 6 3D rendering pipeline

Fig. 7 Typical algorithm run for one frame. *Left*—initialisation, *middle*—intermediate iteration, *right*—final result

inside the triangles. The Z-buffer algorithm is used twice for each pixel in the image, to select the z value for both the closest and the most distant 3D point which project to that 2D pixel.

In the multi-object case multiple renderings are generated: one for each object separately and one with all the objects together—the depth map.

Prisacariu and Reid (2009) implemented a software renderer (so CPU based, rather than GPU based). The software rendering combined with the transfer from host memory to GPU memory accounted for about half of the total processing time per iteration. In this work we implemented a GPU version of the rendering engine, by using a separate thread for each triangle in the model. For the Z-buffer implementation we use global memory atomic write operations to avoid any possible read-write-modify hazards. Note that it is also possible to use one thread per pixel, rather than one thread per triangle. This would mean that each thread would have to check its membership to each triangle, which would quickly become prohibitively expensive as the number of triangles increases. Preprocessing steps could be used to establish which pixels belong to which triangles, but these would also take extra processing time.

In most cases, the GPU implementation is much faster than the CPU implementation. It also scales much better with the number of triangles in the mesh. For example, the model in Fig. 8 (which has 8886 triangles), was rendered in 5.6 ms with the CPU renderer and in 2 ms with the GPU renderer. The only case where the CPU implementation is faster is when model has few triangles (less than 100) but lots of pixels, i.e. its projection is big (more than 100×100). In this case there are considerably more atomic operations than

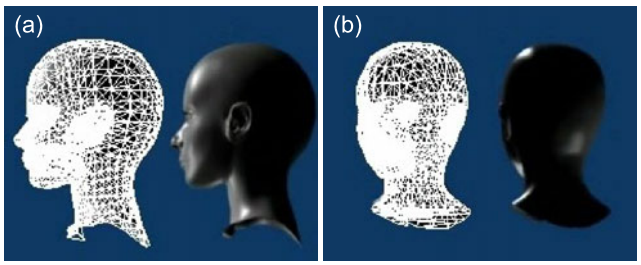

Fig. 8 Failure due to the symmetry of the object. (**a**)—*left*: recovered pose, *right*: rendered frame, (**b**)—*left*: recovered pose, *right*: rendered frame

threads, so the parallelisation is ineffective. Our current implementation chooses dynamically between the two rendering engines. This accounts for the difference in processing time per iteration between that reported in Table 2 and that reported in Table 3, because in the experiments which led to the results shown in Table 2 the algorithm used the CPU renderer, while for those shown in Table 3, the algorithm used the GPU renderer.

5.2 GPU Contour Extraction

We compute the contour of the projection by convolving the projection image with Scharr kernels, and keeping only the pixels which are outside the projection. The Scharr kernels (Scharr 2000) are:

$$\text{Horizontal:} \quad \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Fig. 9 Robustness to motion blur. *Left*—original image, *right*—recovered position

$$\text{Vertical:} \quad \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

This implementation would have been impractical using a sequential algorithm, but it is very parallelisable. Our parallel implementation uses one thread per pixel. In the multi-object case (where a rendering of each object separately is known) the contour is extracted for each object separately.

5.3 GPU Distance Transform

To compute the distance transform we use an algorithm similar to Felzenszwalb and Huttenlocher (2004). 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.

In the multi-object case a distance transform is computed for each object.

5.4 GPU Finite Differences and Energy Function Evaluation

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

6 Results

We tested our method on a variety of models and video sequences. In this section we demonstrate some of the qualities of our method, such as robustness to initialisation, motion blur and occlusions. We also demonstrate the advantage of including the most distant points of the object in the evaluation of the energy function. We showcase the difference between using the pixel-wise posteriors formulation of the level set framework and the classical log likelihood formulation. We compare our method with the one of Brox et al. (2009), which is at or close to the current state of the art in 3D pose tracking. Although Brox et al. (2009) combine region-based tracking with optical flow and SIFT features while we only use the silhouette, we achieve similar or better results, but with a processing time that is hundreds of times smaller. Finally we compare the poses recovered by our method for each frame of a video with the ones obtained by a SLAM (Simultaneous Location and Mapping) system, over the same video.

The source code for our implementation is available online at http://www.robots.ox.ac.uk/ActiveVision.

Figure 7 shows an example of our method performing segmentation on a single frame.[1] Here we are able to successfully recover translational changes in x and y of up to $\pm 40\%$ of the object size, a similar amount in z (scale) and rotational changes of 50 degrees on the x and y axis and 70 degrees on the z axis. Depending on the model and image data, we were able to recover rotations up to 90 degrees on each axis. With regard to translation, the algorithm 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). When the projec-

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

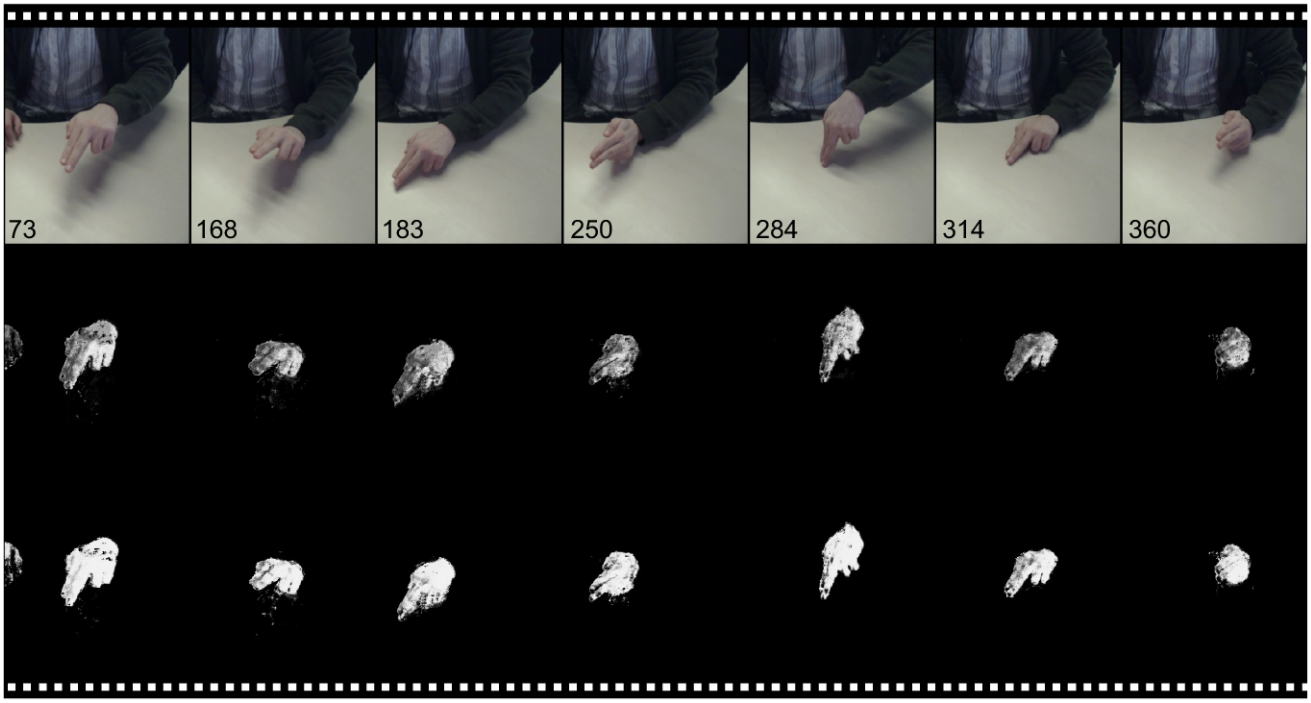

Fig. 10 Effect of temporal consistency on the posterior probabilities. The *top row* shows the original images. In the *second* and *third row*, we plot the per pixel difference $P_f - P_b$ where $P_f - P_b > 0$, scaled to $[0, 255]$. The separation between the foreground and background regions is better when imposing temporal consistency (*third row*)

Fig. 11 Film strip showing our method tracking a hand. Our algorithm can track the hand in this sequence at around 20 fps

tion overlaps with the object in the image, even by a small amount, a pressure develops in the energy function with the effect of increasing this overlap, effectively pulling the object in the right direction.

Figure 8 shows a failure scenario for our method. It is an artificially generated video sequence in which the camera is rotated around a human head. The segmentation is correct but the pose is wrong. This problem arises because the object is symmetrical; the contour alone is insufficient to determine the pose unambiguously. This is of course a limitation of all pose recovery algorithms based solely on the occluding contour.

In standard edge based tracking, because of motion blur, edges often disappear, or have unstable positions. This leads to an advantage of a region-based tracker over an edge based tracker: its robustness to motion blur. In Fig. 9 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 have most likely failed.

Figure 10 shows the effect of imposing temporal consistency on the posterior probabilities. We plot the per pixel difference $P_f - P_b$, where $P_f - P_b > 0$, scaled to $[0, 255]$. Brighter pixels correspond to a larger difference between P_f and P_b. When imposing temporal consistency, the separation between the two regions is better, which leads to a faster and more accurate pose recovery.

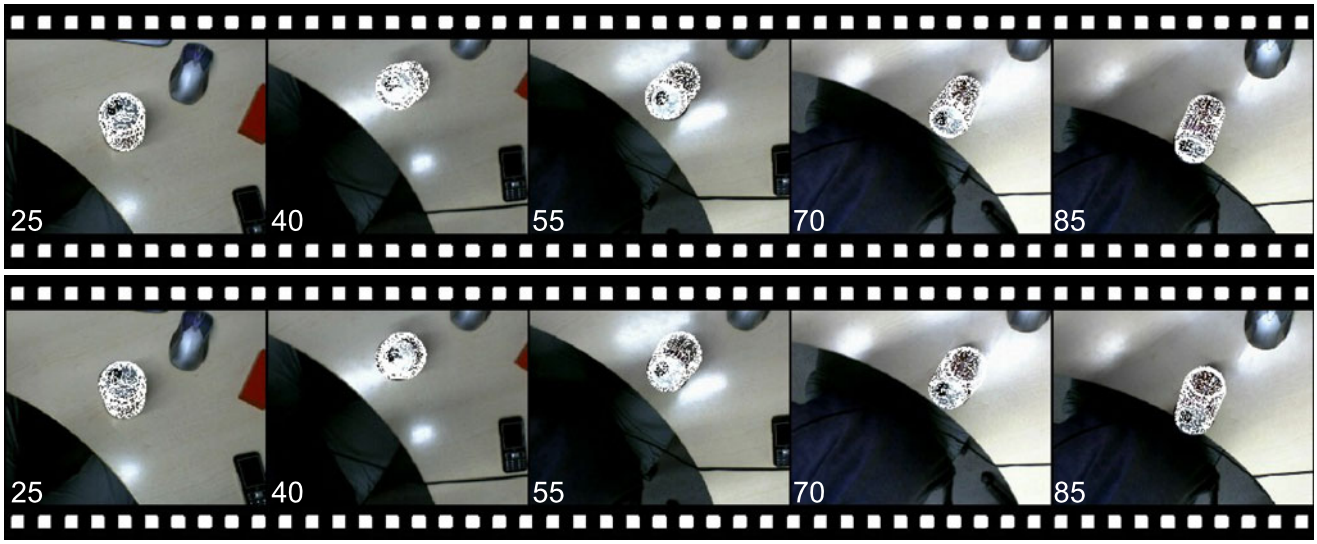

Fig. 12 Film strip showing 5 frames from a video sequence in which the camera moves above a soft-drink can in a circular motion. Between frames 25 and 55 the view of the can moves to directly above, with the side obscured, back to a view with the side visible again. When allowing only the closest points to influence the contour evolution, the pose incorrectly flips (*above*). In contrast the behaviour is correct when the furthest points are also considered (*below*)

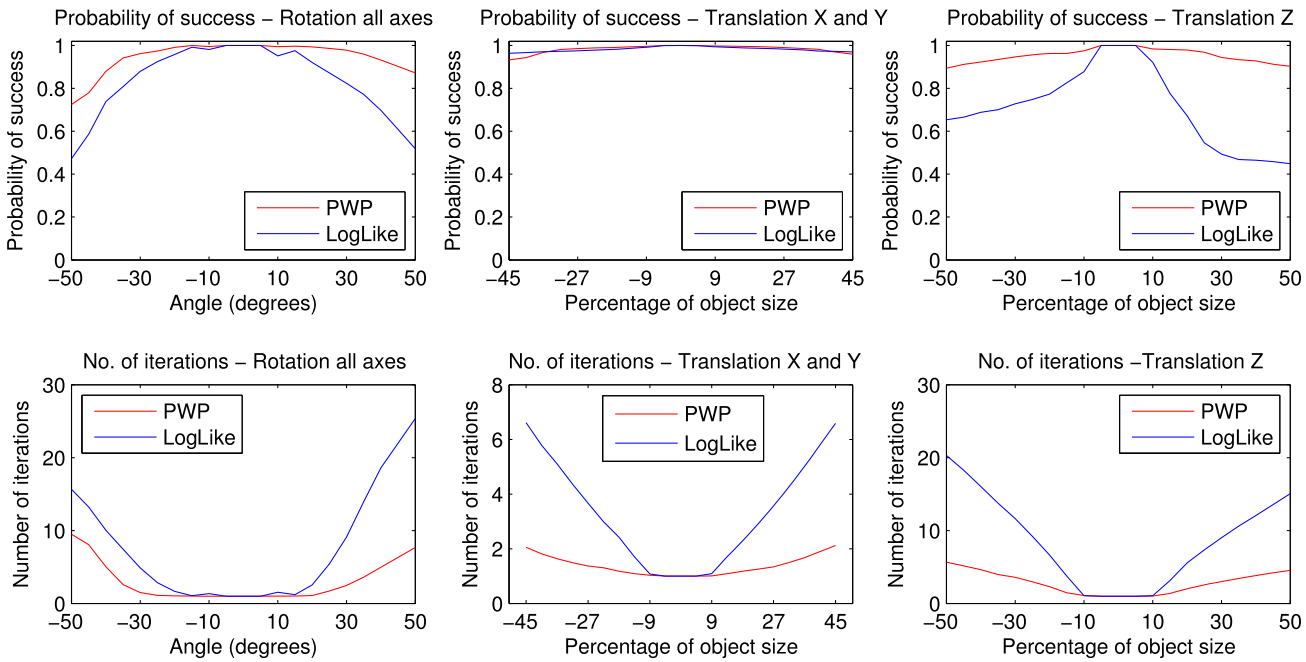

Fig. 13 Performance analysis: probability of convergence to the correct pose generated from over 200,000 experiments

A potential source of problems when using strong prior 3D (or 2D) shape knowledge in tracking and pose recovery are discrepancies between the model and the real target. Figure 11 shows frames from a sequence where we track a human hand with an approximate model. Tracking is still accurate because our method is region-based and probabilistic.

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 in these cases is more difficult. Note that we do not use a motion model, though this could easily be incorporated. This

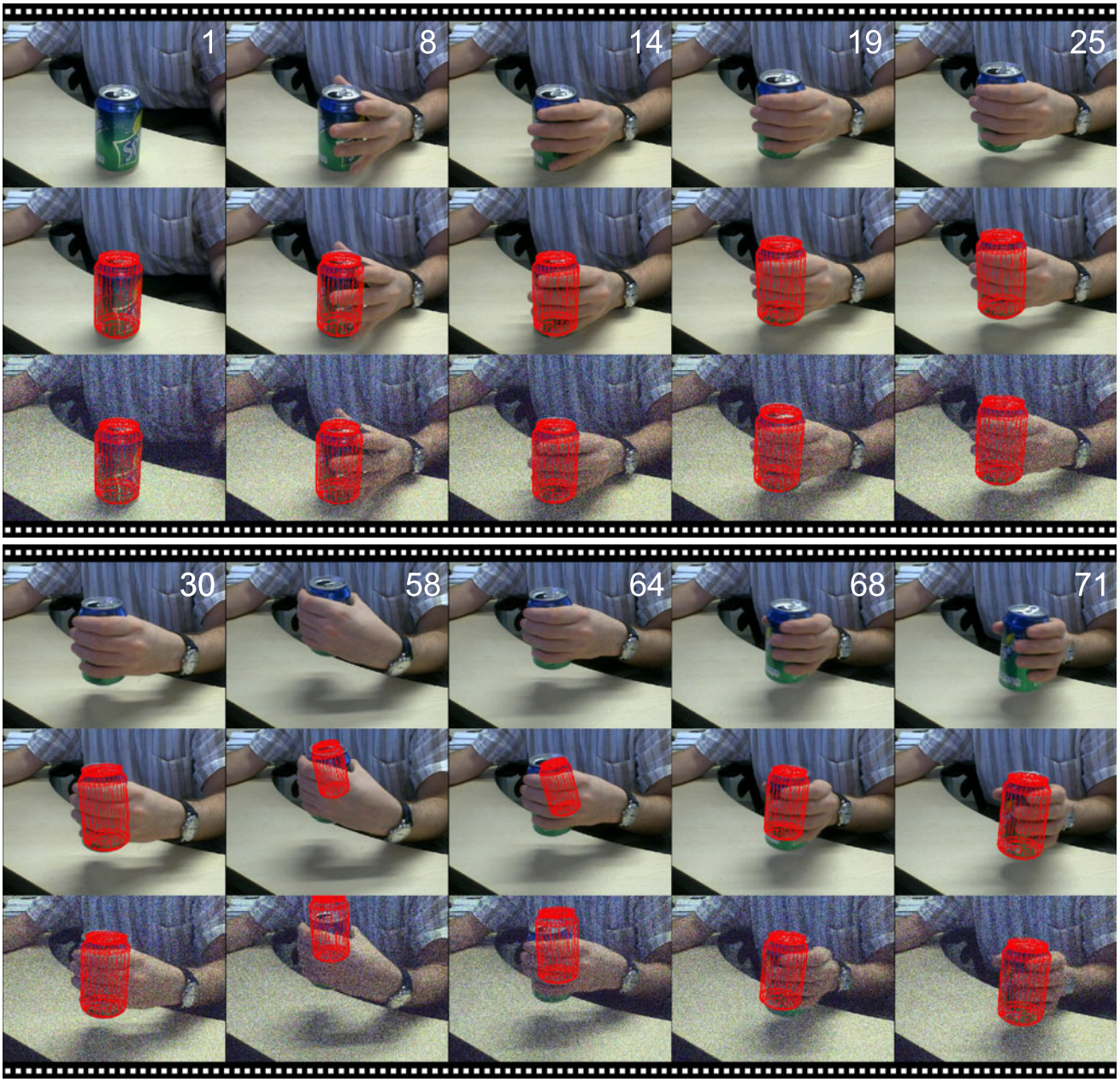

Fig. 14 Film strip consisting of 10 frames from a video which showcases the PWP3D behaviour when subjected to occlusions and high levels of noise. For the first 6 frames (when *both* the *top* and *bottom parts* of the soft-drink can are visible) the pose is correct, even though most of the interior part of the object is occluded. When the *bottom part* of the can becomes occluded the pose becomes incorrect. The object moves "up" in the image because there is no longer a reason for it to stay "down". The algorithm converges back to the correct position once the bottom part stops being occluded

would greatly help in the prediction of the appearance and disappearance of surfaces.

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. As mentioned in Sect. 4, we also take into consideration the most distant points which project to the contour of the projection, in the energy function. Figure 12 shows excerpts 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 allowing only the closest points to influence the contour evolution, the pose incorrectly flips. In contrast the behaviour is correct when the furthest points are also considered.

We used our video sequences to perform a critical performance analysis comparing the performance of both the

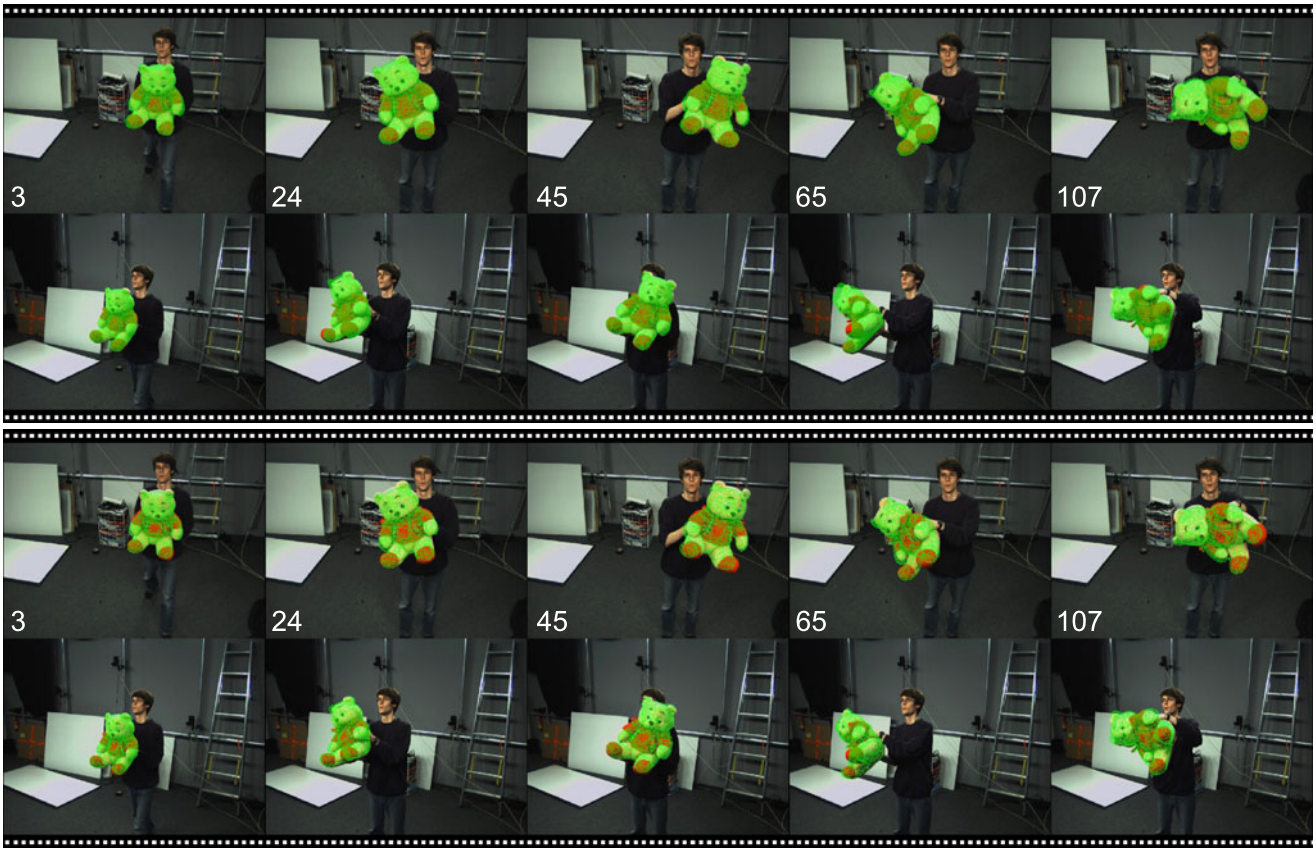

Fig. 15 Comparison between our method (*above*) the one presented by Brox et al. (2009). Although Brox et al. (2009) combine region-based tracking with optical flow and SIFT features to achieve tracking, while we use only the silhouette, our results are very similar, if not better. Note that here we use a single view (*above, first row*), and re-project in the other (*above, second row*), while in Brox et al. (2009) three views are used (from which we included 2 here)

Fig. 16 Filmstrip showing 6 frames from our PTAM comparison video. We used a checker board pattern to calibrate our results with the PTAM (Klein and Murray 2007) ones

pixel-wise posteriors energy function that we use in this paper and the log likelihood one (like that of Cremers et al. (2007) and Dambreville et al. (2008)), for 3D tracking. For each frame we computed 120 perturbations around the correct pose (20 for each pose parameter), in total more than 200,000 experiments. We measured whether each algorithm converged and how many iterations were required for convergence to the true pose,[2] with success defined by a difference of at most 5 degrees in rotation and 5% of object size in translation. Figure 13 summarises 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 axes. Finally, the number of iterations necessary for convergence (when convergence was possible) is considerably higher when using log likelihood (for both rotation and translation). For these experiments we used cuboid objects, like the one in Fig. 4.

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

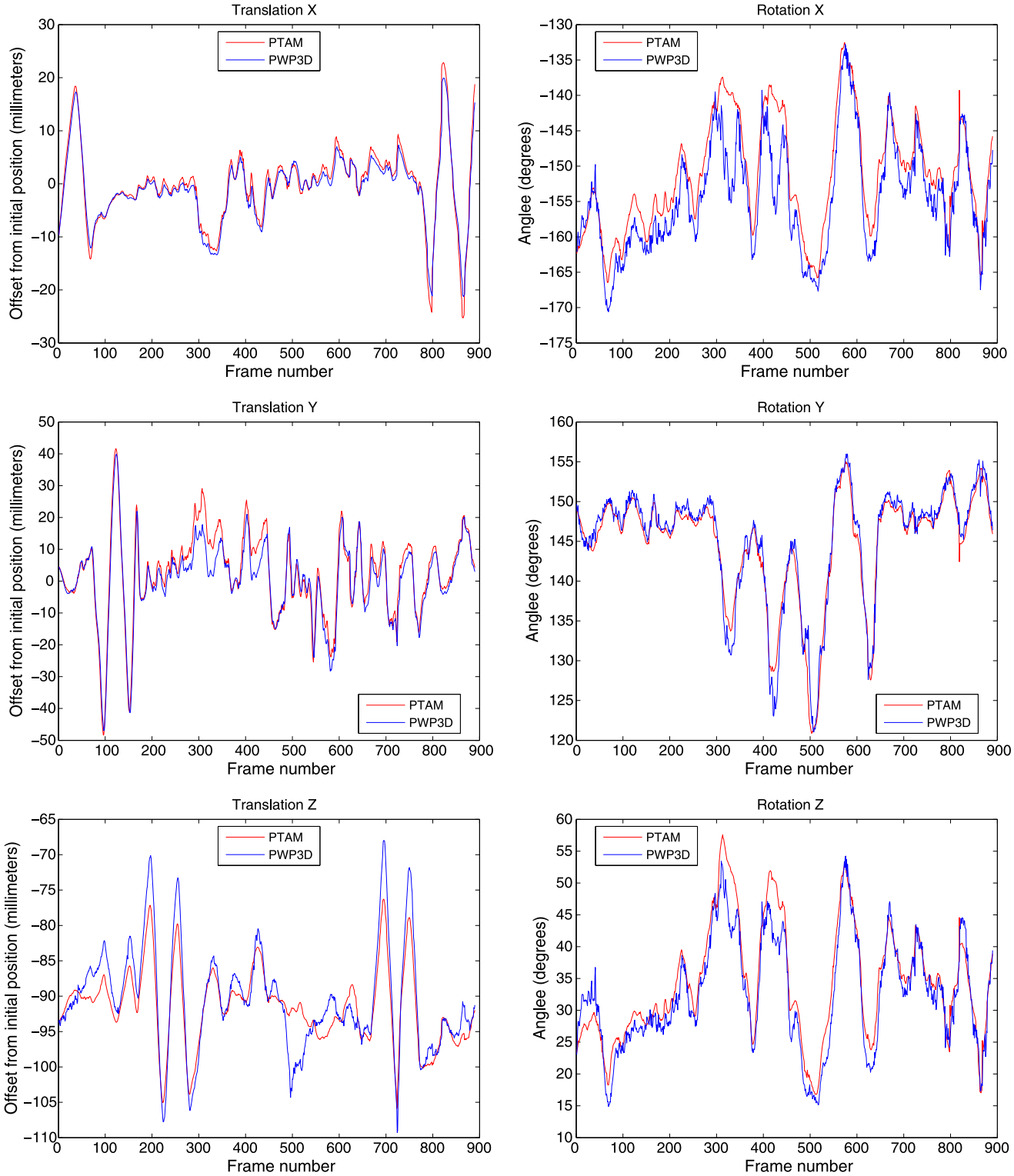

Fig. 17 Comparison between camera pose output when using PTAM (Klein and Murray 2007) and PWP3D

In Fig. 14 we depict a film strip showing the behaviour of PWP3D when subjected to high degrees of occlusion and noise. When both the top and bottom parts of the soft-drink can are visible, the pose is correct, even though most of the interior part of the object is occluded. When the bottom part of the can becomes occluded the pose becomes incorrect. The object moves "up" in the image because there is no longer a reason for it to stay "down". The algorithm

Fig. 18 Multi view tracking. The *top two rows* show the two views. When processing only the first view (*top row*) the results are incorrect (*third row*) because the silhouette to pose mapping is ambiguous (multiple silhouettes can be generated by the same pose). When using two views the ambiguities are eliminated and the results are correct for both views (*bottom two rows*)

converges back to the correct position once the bottom part stops being occluded. Throughout our testing PWP3D has consistently been able to recover from an incorrect pose in no more than 10 frames.

We compared our method with the one proposed by Brox et al. (2009), which is at or close to the current state of the art in 3D model based tracking. The authors use three cues for point correspondences: region-based, dense optical flow-based and SIFT-based. In Fig. 15 we show the results obtained by our method (above) and the one in Brox et al. (2009) (below). The results are very similar, even though we track only the silhouette. Our algorithm is also considerably faster (we can track this object at around 100 ms per frame while the authors report processing a frame in about 80 s). It is important to note that here we use a single view (above, first row), and reproject in the other (above, second row), while in Brox et al. (2009) three views are used (from which we included 2 in Fig. 15). Brox et al. (2009) will of course achieve better results when the pose is more ambiguous (i.e. when more poses project to the same silhouette), but this experiment still demonstrates the efficacy of our method.

Our system obtains the pose parameters for a 3D object relative to a fixed camera. This is the same as obtaining the pose of a 3D camera relative to a stationary object. SLAM systems build a sparse 3D map of the environment and calculate the pose of a 3D camera relative to that map. Therefore we can obtain a measure of accuracy for PWP3D by comparing it against a SLAM system. Neither SLAM nor PWP3D provide the exact pose of the camera but the two systems should converge to very similar results. In Fig. 17 we depict the pose parameters obtained by our system and those obtained by the PTAM (Parallel Tracking and Mapping) system of Klein and Murray (2007), which is at or close to the current state of the art in structure from mo-

Fig. 19 Our algorithm tracking two objects, a soft drink can and a hand. The *top film strip* shows the hand occluding the soft drink can. The *bottom film strip* shows the opposite, the soft drink can occluding the hand. The *third row* in each film strip shows the resulting depth map

tion. In Fig. 16 we show a few frames from the video we used to compare our system with PTAM. The two are very close with the exception of two regions in rotation on X and Z (around frame 350 and 460) and one region on translation in Z (around frame 500). Note that SLAM/PTAM uses many features to obtain the pose of the camera whereas PWP3D uses just the silhouette. The differences in rotation are caused by the fact that we are tracking only the silhouette and, in those regions, the object was viewed from the side. A tilt in the X or Z direction had no effect on the silhouette. The difference in translation on Z is caused by a high degree of motion blur, which corrupted the features used by PTAM.

In Fig. 18 our algorithm uses two views to disambiguate the pose of a hand. The top two rows show the two views. The straight on view (top row) of the hand is ambiguous i.e. multiple poses can generate the same silhouette. Row three shows the pose obtained by our algorithm when processing only the first view (top row). The algorithm is confused by the ambiguity in the pose-silhouette function and converges to a wrong result. When adding the second view (second row) the pose-silhouette function is disambiguated

and the pose is recovered successfully in both views (last two rows). Processing time suffers considerably, as processing two views is twice as slow as using a single view. Multiple threads and/or video cards (one for each view) could be used to speed up the algorithm.

Figure 19 shows our algorithm tracking two objects, a soft drink can and a hand. In the top film strip the hand occludes the soft drink can, while in the second film strip the soft-drink hand occludes the hand. Tracking is consistent in both cases. Processing time does again suffer considerably. Tracking two objects is about 2.5 times slower than tracking a single object, two times slower because two energy functions are computed, plus the extra processing required for generating and using the depth map.

7 Conclusions

In this article we have described a novel model-based 3D tracker, constructed on the assumption that the segmentation of an object in an image is fully defined by the 6 DoF pose space of its known 3D model. The method is based on the computation of the derivatives of a level set based segmentation energy function with respect to the pose parameters of the known 3D model. Segmentation and pose are then recovered by standard nonlinear optimisation techniques, such as gradient descent and conjugate gradients, resulting in a fully variational, probabilistic, one shot, level set minimisation. Inspired by Bibby and Reid (2008) we have adopted the use of pixel-wise posteriors (in contrast to likelihoods) and shown that, as in the 2D case addressed by Bibby and Reid (2008), we achieve performance gains also in the 3D case. In addition, we have introduced further improvements at the pixel level such as consideration of temporal consistency and improved online adaptation of foreground and background appearance models. Importantly, we have also demonstrated that, because of its highly parallel nature, our algorithm is amenable to real-time implementation using GPU acceleration, making this one of the few model and region based 3D trackers to achieve real time performance. Our method shares the benefits of region based segmentation with shape priors, such as robustness to occlusion, noise or motion-blur, while still allowing for large changes in rotation and translation to be successfully recovered, and these aspects have been experimentally demonstrated. We have shown how straightforward extensions lead to the ability to track multiple objects from multiple views, which overcomes ambiguities created by the fact that we only use silhouette information in finding the 6D pose.

Though we have made some improvements to the online appearance model adaptations allowing for more aggressive updating, we are still only using a single, global appearance model of the tracked object. More interestingly, one could take advantage of the fact that appearance of a 3D object is a function of pose, and we believe that (online) learning of such an appearance model may be both feasible and beneficial. Several techniques also exist for learning manifolds of 2D shapes by use of dimensionality reduction techniques, an example being our work on shape spaces of Prisacariu and Reid (2011a) and shared shape spaces of Prisacariu and Reid (2011b). By combining our algorithm with such a technique, we could potentially find not only the 3D pose of the object, but also its shape, and such an idea may well extend to articulated and/or deformable objects. Tracking articulations would also be possible by extending our method in the spirit of Kohli et al. (2008), combined with a multi view formulation such as the one of Liu et al. (2011).

Acknowledgements We thank Bodo Rosenhahn for sharing the videos, 3D models and results for the paper Brox et al. (2009). We are grateful for discussions with Charles Bibby. Victor Adrian Prisacariu gratefully acknowledges the financial support of the EPSRC and Balliol College, University of Oxford.

References

Bibby, C., & Reid, I. (2008). Robust real-time visual tracking using pixel-wise posteriors. In *ECCV 2008* (pp. 831–844).

Binford, T. O. (1981). Inferring surfaces from images. *Artificial Intelligence*, *17*(1-3), 205–244.

Bouguet, J. Y. (2008). Camera calibration toolbox for Matlab.

Brox, T., Rosenhahn, B., Gall, J., & Cremers, D. (2009). Combined region- and motion-based 3D tracking of rigid and articulated objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(3), 402–415.

Cremers, D., Osher, S. J., & Soatto, S. (2006). Kernel density estimation and intrinsic alignment for shape priors in level set segmentation. *International Journal of Computer Vision*, *69*(3), 335–351.

Cremers, D., Rousson, M., & Deriche, R. (2007). A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape. *International Journal of Computer Vision*, *72*(2), 195–215.

Dambreville, S., Sandhu, R., Yezzi, A., & Tannenbaum, A. (2008). Robust 3D pose estimation and efficient 2D region-based segmentation from a 3D shape prior. In *ECCV 2008* (pp. 169–182).

Drummond, T., & Cipolla, R. (1999). Visual tracking and control using Lie algebras. In *CVPR 1999* (pp. 652–657).

Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Distance transforms of sampled functions. Tech. Rep., Cornell Computing and Information Science.

Freeman, W. T. (1993). Exploiting the generic view assumption to estimate scene parameters. In *ICCV 1993* (pp. 347–356).

Gall, J., Rosenhahn, B., & Seidel, H. P. (2008). Drift-free tracking of rigid and articulated objects. In *CVPR 2008* (pp. 1–8).

Gilbert, J. C., & Nocedal, J. (1992). Global convergence properties of conjugate gradient methods for optimization. *SIAM Journal on Optimization*, *2*(1), 21–42.

Hager, G., & Belhumeur, P. (1998). Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(10), 1025–1039.

Hager, W. W., & Zhang, H. (2005). A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on Optimization*, *16*(1), 170–192.

Harris, C. (1993). Tracking with rigid models. In *Active vision* (pp. 59–73).

Jurie, F., & Dhome, M. (2002). Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(7), 996–1000.

Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *ISMAR 2007* (pp. 1–10).

Kohli, P., Rihan, J., Bray, M., Torr, P. H. S., Kohli, P., Rihan, J., Bray, M., & Torr, P. H. S. (2008). Simultaneous segmentation and pose estimation of humans using dynamic graph cuts. In *IJCV* (pp. 285–298).

Lepetit, V., & Fua, P. (2005). Monocular model-based 3D tracking of rigid objects: a survey. *Foundations and Trends in Computer Graphics and Vision*, *1*(1), 1–89.

Lepetit, V., Lagger, P., & Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *CVPR 2005* (pp. 775–781).

Liu, Y., Stoll, C., Gall, J., Seidel, H. P., & Theobalt, C. (2011). Markerless motion capture of interacting characters using multi-view image segmentation. In *CVPR 2011* (pp. 1249–1256).

Lowe, D. G. (1992). Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, *8*(2), 113–122.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.

Marchand, E., Bouthemy, P., Chaumette, F., & Moreau, V. (1999). Robust real-time visual tracking using a 2D–3D model-based approach. In *ICCV 1999* (pp. 262–268).

NVIDIA (2009). NVIDIA CUDA Programming Guide 2.2.

Osher, S., & Sethian, JA (1988). Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, *79*(1), 12–49.

Ozuysal, M., Lepetit, V., Fleuret, F., & Fua, P. (2006). Feature harvesting for tracking-by-detection. In *ECCV 2006* (pp. 592–605).

Press, W. H. (2007). *Numerical recipes: the art of scientific computing* (3rd edn.). Cambridge: Cambridge University Press.

Prisacariu, V., & Reid, I. (2009). PWP3D: real-time segmentation and tracking of 3D objects. In *BMVC 2009* (pp. 1–10).

Prisacariu, V., & Reid, I. (2011a). Nonlinear shape manifolds as shape priors in level set segmentation and tracking. In *CVPR 2011* (pp. 2185–2192).

Prisacariu, V., & Reid, I. (2011b). Shared shape spaces. In *ICCV 2011*.

Rosenhahn, B., Brox, T., & Weickert, J. (2007). Three-dimensional shape knowledge for joint image segmentation and pose tracking. *International Journal of Computer Vision*, *73*(3), 243–262.

Rosten, E., & Drummond, T. W. (2005). Fusing points and lines for high performance tracking. In *ICCV 2005* (pp. 1508–1511).

Scharr, H. (2000). Optimal operators in digital image processing. PhD thesis.

Schmaltz, C., Rosenhahn, B., Brox, T., Cremers, D., Weickert, J., Wietzke, L., & Sommer, G. (2007a). Region-based pose tracking. In *IbPRIA 2007* (pp. 56–63).

Schmaltz, C., Rosenhahn, B., Brox, T., Weickert, J., Cremers, D., Wietzke, L., & Sommer, G. (2007b). Occlusion modeling by tracking multiple objects. In *DAGM 2007* (pp. 173–183).

Vese, L. A., & Chan, T. F. (2002). A multiphase level set framework for image segmentation using the Mumford and shah model. *International Journal of Computer Vision*, *50*(3), 271–293.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., & Schmalstieg, D. (2008). Pose tracking from natural features on mobile phones. In *ISMAR 2008* (pp. 125–134).