Real-Time Monocular Segmentation and Pose Tracking of Multiple Objects

Henning Tjaden[1], Ulrich Schwanecke[1], and Elmar Schömer[2]

[1]Computer Science Department, RheinMain University of Applied Sciences
[2]Institute of Computer Science, Johannes Gutenberg University Mainz
{henning.tjaden,ulrich.schwanecke}@hs-rm.de, schoemer@uni-mainz.de

Abstract. We present a real-time system capable of segmenting multiple 3D objects and tracking their pose using a single RGB camera, based on prior shape knowledge. The proposed method uses twist-coordinates for pose parametrization and a pixel-wise second-order optimization approach which lead to major improvements in terms of tracking robustness, especially in cases of fast motion and scale changes, compared to previous region-based approaches. Our implementation runs at about 50–100 Hz on a commodity laptop when tracking a single object without relying on GPGPU computations. We compare our method to the current state of the art in various experiments involving challenging motion sequences and different complex objects.

Keywords: Tracking, Segmentation, Real-Time, Monocular, Pose Estimation, Model-based, Shape Knowledge

1 Introduction

Tracking the 3D motion of a rigid object from its 2D projections into image sequences of a single camera is one of the main research areas in computer vision. This involves estimating the pose, i.e. the 3D translation and rotation, of the object relative to the camera in each image. The fields of application for visual 3D object tracking are numerous, such as visual servoing of robots, medical navigation and visualization, sports therapy, augmented reality systems and human computer interaction. Many different solutions to this problem have been developed over the years and are now part of a variety of practical applications. For a survey of monocular 3D tracking of rigid objects see e.g. [1].

Recently so-called region-based pose estimation methods have emerged, which are mainly based on statistical level-set segmentation approaches [2]. These do not require any kind of artificial marker or other augmentation of the object of interest. They only rely on a 3D model of the object. Therefore, they fall into the category of model-based pose estimation methods. These are very attractive for application scenarios where it is undesirable or even impossible to modify the objects. For example in case of sports therapy, where ergonomics might be affected or for tracking objects in public environments such as cars or planes that cannot be modified beforehand.

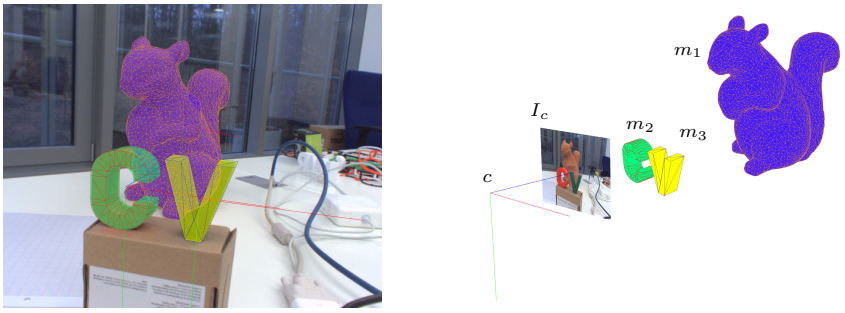

Fig. 1. Example of region-based 3D tracking of three different objects with partial occlusions. All poses are determined by the proposed method within about 30 ms. Left: Augmented reality view of the scene, where the rendered models yield a segmentation of the objects in the image. Right: 3D overview of the scene.

Some model-based approaches use edge or point features associated with the surface model of the 3D object for pose estimation [3–6]. The main disadvantages of these methods are that they struggle with motion blur and are prone to local minima especially with cluttered backgrounds. Using point-based features also requires the objects' surfaces to be sufficiently textured, which significantly limits the variety of suitable objects.

In contrast to feature-based approaches, region-based methods minimize the discrepancy between silhouettes. Thereby, the object's contour is estimated and its 3D pose is determined in an interleaved manner. Here the model is used to generate synthetic projections of the object's silhouette under a currently estimated pose. For pose estimation the similarity between this silhouette and the silhouette stemming from the currently segmented object region in the real image is maximized by varying the pose parameters. Vice versa, the synthetic silhouette which is rendered using an estimated pose is assumed to provide a ground truth segmentation for the current image that is used in order to update the foreground and background image segmentation model (Figure 1).

1.1 Related Work

Due to the large amount of literature about optical pose estimation and tracking, in the following we focus on the pose optimization strategies and real-time capabilities of region-based methods. One class of object pose estimation approaches that couple image segmentation and pose estimation is based on explicit point correspondences between the segmented and the projected contour of the object [7–9]. These correspondences are used to set up a linear system describing the spatial distance between the projection rays of the points on the segmented contour and the corresponding 3D object points that belong to the projected contour with respect to the pose parameters. The linear system is then solved iteratively by recalculating the point correspondences after each iteration. Over the years, region-based methods involving this point-based pose optimization

procedure, which can be seen as an ICP (iterative closest points) algorithm, have been improved [10, 11] and shown to produce promising results in many challenging scenarios. Unfortunately, the overall computations take several seconds per frame. In [8] an optical flow approach matching the contours by minimizing the pixel-wise quadratic difference between their level-set functions was introduced which determines the object's pose with 13 frames per second for small images.

In [12] the authors present PWP3D, the first region-based approach that achieved real-time frame rates (20–25 Hz) using GPUs by solving the pose estimation problem in a pixel-wise minimization scheme. The presented approach is similar to the variational approach suggested in [13] but uses level-set functions instead of separately integrating over the foreground and background region to simplify computations and make it real-time capable. More recently, a further improved version of PWP3D was presented, that runs with more than 30 Hz on a mobile phone [14]. The performance increase was mainly due to an hierarchical approach, an approximation of the level set transform and its derivatives, as well as an estimation of the rotation using a gyroscope which is visually corrected for drift based on a single gradient descent step every tenth frame. The authors state that solving the problem in a fully analytical manner without any approximations is only possible with about 20–25 Hz, even when using GPUs.

Other approaches building upon PWP3D have been proposed in [15, 16]. They mainly focus on improving the segmentation part but also suggest different strategies for replacing the simple gradient descent approach. In [15] the Levenberg-Marquardt algorithm is suggested but neither elaborately illustrated nor evaluated with regard to runtime. Gradient descent with two different step sizes, one for rotation and one for translation, is used in [16]. The two gradients are normalized to unit rotation and unit pixel size and a 2D search for the optimal step sizes is conducted. The authors' implementation is much slower than PWP3D but they predict that their algorithm could be further optimized in order to achieve similar runtimes to PWP3D. Both of these methods as well as [13] and [14] use a decoupled rotation and translation vector for pose parametrization.

1.2 Motivation

To our best knowledge PWP3D is still the state of the art monocular region-based pose estimation approach achieving fairly high frame rates. Therefore, we built upon PWP3D and improved it with regard to computation time and tracking robustness. We identified two main areas of potential improvement, image segmentation and pose optimization strategy. Issues resulting from the global foreground and background statistics used for image segmentation have recently been addressed by [15, 16]. As the suggested improvements are not yet real-time capable, we are not using them in our current implementation. In fact, our work disregards the image segmentation part and focusses on improving the pose optimization part to improve the general performance and robustness.

The PWP3D implementation provided by the authors uses a simple gradient descent with predefined, decreasing step sizes in order to minimize the defined

energy function. The number of iterations and the choice of three different step sizes (one each for rotation, translation along the optical axis and translation within the camera's image plane) depend on the model complexity and the size of the projected silhouette. Since these parameters have to be adapted at least once for each new object the algorithm is difficult to configure. In general this problem can not be solved by simply using smaller step sizes as real-time frame rates are only achieved with about eight iterations per frame and the number of necessary gradient descent iterations increases if the step sizes decrease. Although others have developed strategies for improving the simple gradient descent in their works where PWP3D was used, to our knowledge the problem has not yet been adequately investigated or improved regarding tracking robustness and runtime reduction for real-time scenarios.

1.3 Contributions

This paper includes three main contributions. First, we present a novel pixel-wise optimization strategy based on a Gauß-Newton-like algorithm using linearized twists for pose parametrization. Our approach has significantly improved convergence properties, especially for rotational motion, as we demonstrate in our experiments. The number of iterations and the common step size can thus be adjusted regardless of the model complexity and its distance to the camera. Second, we describe an implementation that uses the GPU only for rendering purposes, performing all other computations on the CPU and achieving frame rates of about 50–100 Hz when tracking a single object on a commodity laptop. Finally, we present real-time multiple object tracking using region-based pose estimation.

The rest of the paper is structured as follows: Section 2 gives an overview of the used mathematical concepts. Section 3 details our implementation and runtime analysis. A comparison of our approach and previous state of the art based on various experiments is given in Section 4, whereas Section 5 concludes with a brief summary and potential future work.

2 Method

In the following we give a condensed overview of the mathematical concepts and notations used throughout this paper. A camera color image is denoted by $I_c : \Omega \to \mathbb{R}^3$ where $\Omega \subset \mathbb{R}^2$ is the image domain (see Figure 1). The color of each pixel $\mathbf{x}_c := (x_c, y_c)^\top \in \Omega$ is given by $\mathbf{y} = I_c(\mathbf{x}_c)$. Each object is represented by a dense surface model (triangle mesh) consisting of 3D points $\mathbf{X}_m^i := (X_m^i, Y_m^i, Z_m^i)^\top \in \mathbb{R}^3$, with $i = 1, \ldots, N$ where N is the number of objects. The pose of a model m_i with respect to the camera coordinate frame c is given by

$$T_{cm}^i = \begin{bmatrix} R_{cm}^i & \mathbf{t}_{cm}^i \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in \mathbb{SE}(3), \tag{1}$$

describing the rigid transformation between surface points $\mathbf{X}_m^i$ in the model's reference frame and $\mathbf{X}_c^i$ in the camera's reference frame by $\tilde{\mathbf{X}}_c^i = T_{cm}^i \tilde{\mathbf{X}}_m^i$. The

tilde denotes the corresponding homogenous representation of a vector $\tilde{\mathbf{X}} = (X, Y, Z, W)^\top$, with $W = 1$. $R^i_{cm} \in \mathbb{SO}(3)$ denotes the model's orientation and $\mathbf{t}^i_{cm} \in \mathbb{R}^3$ the model's origin relative to the camera's reference frame.

For pose optimization we represent the rigid body motion that occurred between two consecutive frames using twists

$$\theta\hat{\xi} = \theta \begin{bmatrix} \hat{\mathbf{w}} & \mathbf{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3), \text{ with } \hat{\mathbf{w}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in \mathfrak{so}(3), \tag{2}$$

which are parametrized by the vector $\theta\xi = \theta\left(\omega_1, \omega_2, \omega_3, v_1, v_2, v_2\right)^\top \in \mathbb{R}^6$, with $\mathbf{w} = (w_1, w_2, w_3)^\top$, $||\mathbf{w}||_2 = 1$, where θ is a one-parametric coupling of the rotation and translation parameters describing the motion along a screw. This coupling qualifies twist coordinates as the natural choice for gradient-based pose optimization methods. For more information on the *Lie group* $\mathbb{SE}(3)$ and the corresponding *Lie algebra* $\mathfrak{se}(3)$ see [17]. Each twist can be transformed to its corresponding group element via the exponential map

$$exp(\theta\hat{\xi}) = \begin{bmatrix} exp(\theta\hat{\mathbf{w}}) & (\mathbb{I}_{3\times 3} - exp(\theta\hat{\mathbf{w}}))\hat{\mathbf{w}}\mathbf{v} + \mathbf{w}\mathbf{w}^\top\mathbf{v}\theta \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in \mathbb{SE}(3), \tag{3}$$

where $exp(\theta\hat{\mathbf{w}})$ can be computed according to Rodrigues's formula as

$$exp(\theta\hat{\mathbf{w}}) = \mathbb{I}_{3\times 3} + sin(\theta)\hat{\mathbf{w}} + (1 - cos(\theta))\hat{\mathbf{w}}^2. \tag{4}$$

Our camera is pre-calibrated and its intrinsic matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

is assumed to be fixed. All images are undistorted removing non-linear distortion and allowing the perspective projection of a 3D surface point to a 2D image point to be described by $\mathbf{x}_c = \pi(K(T_{cm}\tilde{\mathbf{X}}_m)_{3\times 1})$, with $\pi(\mathbf{X}) = (X/Z, Y/Z)^\top$.

Our method builds upon PWP3D where the pixel-wise posterior probability of a projected contour, given the current camera image is described by

$$P(\Phi^i|I) = \prod_{\mathbf{x}_c \in \Omega} \Big(H_e(\Phi^i(\mathbf{x}_c))P^i_f(\mathbf{y}) + (1 - H_e(\Phi^i(\mathbf{x}_c)))P^i_b(\mathbf{y}) \Big). \tag{6}$$

$P^i_f(\mathbf{y})$ and $P^i_b(\mathbf{y})$ represent the per object foreground and background region membership probability of each pixel's color, as illustrated in Section 2.1. H_e is the smoothed Heaviside step function

$$H_e(x) = \frac{1}{\pi}\left(-\arctan(b \cdot x) + \frac{\pi}{2}\right), \tag{7}$$

with b determining its reach. Φ^i is the level-set embedding of each object's contour defined by its pose as described in Section 2.2. Assuming pixel-wise independency and by taking the negative log of $P(\Phi|I)$, this results in the energy

$$E^i = -\sum_{\mathbf{x}_c \in \Omega} \log\Big(H_e(\Phi^i(\mathbf{x}_c))P^i_f(\mathbf{y}) + (1 - H_e(\Phi^i(\mathbf{x}_c)))P^i_b(\mathbf{y}) \Big), \tag{8}$$

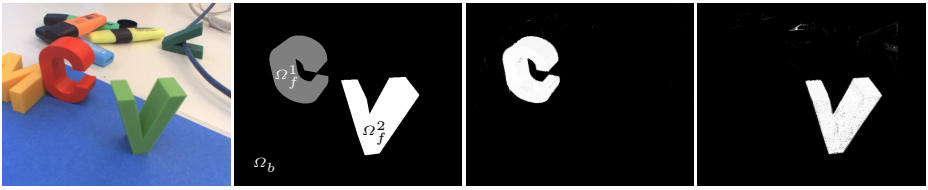

Fig. 2. Pixel-wise posterior image segmentation. From left to right: Camera image of two 3D objects to be segmented: A red C and a green V. Silhouettes of corresponding surface models drawn in a grey value corresponding to its index. Per pixel difference $P_f^1(\mathbf{y}) - P_b^1(\mathbf{y})$ and $P_f^2(\mathbf{y}) - P_b^2(\mathbf{y})$ respectively, visualizing the resulting segmentation

to be minimized with respect to the respective object's pose. Our approach to efficiently perform this minimization is described in Section 2.3.

2.1 Pixel-wise Posterior Object Segmentation

We are basically using the same image segmentation method as PWP3D, based on the pixel-wise posteriors presented in [18] (see Figure 2). Assuming that the pose of each object is known for the current image, their 3D models are used to render synthetic silhouettes splitting the image domain in multiple foreground regions Ω_f^i and a common background region Ω_b. Thus the individual background regions are given by $\Omega_b^i = \Omega \setminus \Omega_f^i$. These regions are used to determine foreground and background RGB color histograms with 32 bins per channel modelling the foreground and background posteriors $P_f^i(\mathbf{y})$ and $P_b^i(\mathbf{y})$ for each color. We are also using PWP3D's temporal consistency strategy for histogram updating.

2.2 Level-Set Pose Embedding

Each object projects to a silhouette region Ω_f^i, that depends on the object's pose and is bounded by its 2D contour $\mathbf{C}^i$ (see Figure 3). A level-set function Φ implicitly defines such a closed curve as its zero level. Here Euclidian signed distance transforms are used to define this level-set embedding as

$$\Phi^i(\mathbf{x}_c) = \begin{cases} -d(\mathbf{x}_c, \mathbf{C}^i) & \forall \mathbf{x}_c \in \Omega_f^i \\ d(\mathbf{x}_c, \mathbf{C}^i) & \forall \mathbf{x}_c \in \Omega_b^i \end{cases}, \quad \text{with} \quad d(\mathbf{x}_c, \mathbf{C}) = \min_{\mathbf{c} \in \mathbf{C}} |\mathbf{c} - \mathbf{x}_c|, \tag{9}$$

where each pixel is mapped to the shortest distance between its location and the contour (see Figure 3). Such signed distance transforms can be efficiently computed with the method presented in [19]. By a simple extension, this algorithm can also be used to compute a mapping of each pixel to the corresponding location of its closest contour pixel, which is needed during pose optimization, as explained in the following Section.

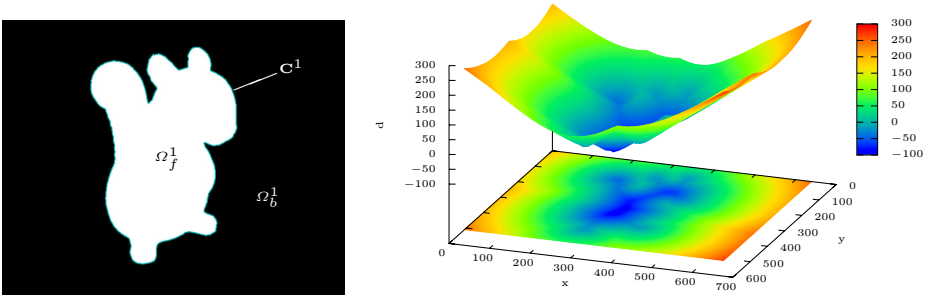

Fig. 3. An illustration of the level-set pose embedding $\Phi(\mathbf{x}_c)$ applied to a squirrel figurine. Left: The projected silhouette with indicated contour $\mathbf{C}^1$ as well as inner region Ω_f^1 and outer region Ω_b^1. Right: A combined 2D/3D plot of the signed distance transform of $\mathbf{C}^1$

2.3 Iterative Pose Optimization

In contrast to the originally applied gradient descent approach, we optimize the energy functional (Eq. 8) with respect to the pose parameters based on an iterative Gauß-Newton-like method. For nonlinear optimization methods the number of required gradient calculations and function evaluations has the greatest impact on runtime. In our situation rendering a silhouette and computing its signed distance transform are quite costly operations making a function evaluation similarly computationally demanding as calculating its gradient. We therefore neglected optimization methods using any kind of line-search strategy to determine step sizes, such as conjugate gradients, DFP, BFGS or Levenberg-Marquardt [20]. Calculating the Hessian matrix in order to determine the step size can be done with an acceptable increase of runtime, but our experiments proved it to be numerically unstable, as commonly reported in literature.

We instead selected an iterative Gauß-Newton-like optimization strategy, where the Hessian is approximated from first order derivatives. Thereby, the gradient of the energy functional is given by

$$\frac{\partial E^i(\theta\xi)}{\partial\theta\xi} = -\sum_{\mathbf{x}_c\in\Omega} \frac{P_f(\mathbf{y}) - P_b(\mathbf{y})}{H_e(\Phi^i(\mathbf{x}_c,\theta\xi))(P_f(\mathbf{y}) - P_b(\mathbf{y})) + P_b(\mathbf{y})}\delta_e \frac{\partial\Phi^i(\mathbf{x}_c,\theta\xi)}{\partial\theta\xi} \tag{10}$$

where δ_e is the smoothed Dirac delta function corresponding to H_e and the derivatives of the signed distance function with respect to twist coordinates are

$$\frac{\partial\Phi^i(\mathbf{x}_c,\theta\xi)}{\partial\theta\xi} = \frac{\partial\Phi^i}{\partial\mathbf{x}_c}\frac{\partial\pi}{\partial K(exp(\theta\hat{\xi})T^i_{cm}\tilde{\mathbf{X}}^i_m)_{3\times1}}\frac{\partial K(exp(\theta\hat{\xi})T^i_{cm}\tilde{\mathbf{X}}^i_m)_{3\times1}}{\partial\theta\xi}. \tag{11}$$

Assuming small motion it holds $exp(\theta\hat{\xi}) \approx \mathbb{I}_{4\times4} + \theta\hat{\xi}$ and therefore we get

$$\frac{\partial\Phi^i(\mathbf{x}_c,\theta\xi_0)}{\partial\theta\xi} = \left[\frac{\partial\Phi^i}{\partial x_c}, \frac{\partial\Phi^i}{\partial y_c}\right] \begin{bmatrix} \frac{f_x}{Z_c^i} & 0 & -\frac{X_c^i f_x}{(Z_c^i)^2} \\ 0 & \frac{f_y}{Z_c^i} & -\frac{Y_c^i f_y}{(Z_c^i)^2} \end{bmatrix} \begin{bmatrix} 0 & Z_c^i & -Y_c^i & 1 & 0 & 0 \\ -Z_c^i & 0 & X_c^i & 0 & 1 & 0 \\ Y_c^i & -X_c^i & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{12}$$

with $\mathbf{X}_c^i = (X_c^i, Y_c^i, Z_c^i)^\top = (T_{cm}^i \tilde{\mathbf{X}}_m^i)_{3\times 1}$. For a pixel $\mathbf{x}_c \in \Omega_b^i$, $\mathbf{X}_c$ is the surface point in the camera's frame of reference that projects to its closest contour pixel. The first-order derivatives of Φ^i are simply calculated as central differences

$$\frac{\partial \Phi^i}{\partial x_c} = \frac{\Phi^i(x_c+1, y_c) - \Phi^i(x_c-1, y_c)}{2}, \quad \frac{\partial \Phi^i}{\partial y_c} = \frac{\Phi^i(x_c, y_c+1) - \Phi^i(x_c, y_c-1)}{2}. \quad (13)$$

This results in the 1×6 per pixel Jacobi vector $J^i(\mathbf{x}_c, \theta\xi_0) = \partial E^i(\mathbf{x}_c, \theta\xi_0)/\partial\theta\xi$, evaluated at $\theta\xi_0 = \mathbf{0}^\top$. At each iteration the twist parameter step is calculated as

$$\Delta\theta\xi^i = -\left(\sum_{\mathbf{x}_c \in \Omega} J^i(\mathbf{x}_c, \theta\xi_0)^\top J^i(\mathbf{x}_c, \theta\xi_0)\right)^{-1} \sum_{\mathbf{x}_c \in \Omega} J^i(\mathbf{x}_c, \theta\xi_0)^\top, \quad (14)$$

using Cholesky decomposition. The resulting step is mapped to its corresponding group element in $\mathbb{SE}(3)$ and applied to the initial transform estimate as

$$T_{cm}^i \leftarrow exp(\Delta\theta\xi^i) T_{cm}^i. \quad (15)$$

In order to increase tracking robustness in case of fast movement or motion blur and to decrease runtime, we compute the pose optimization in a coarse to fine manner as explained in detail in Section 3.2.

2.4 Initialization

The whole tracking process is currently initialized manually. We render each model as an overlay to the live camera feed using a pre-defined initial pose. The user is then required to roughly align this mask with the objects in the scene. On key press these silhouette masks along with the current camera image are used to calculate the initial color histograms that provide posterior segmentation for the subsequent first pose optimization. After initialization pose optimization and color histogram updates are performed in an interleaved fashion for each following camera image as described above.

3 Implementation

In the following we provide an overview of our C++ implementation with regard to runtime optimization. It is based on CPU parallelization and OpenGL for rendering purposes.

3.1 Rendering Engine

As PWP3D we use a mesh representation of the 3D surface models. In contrast to PWP3D we do not involve a reverse Z-buffer since it did not show any significant advantage in our experiments. Instead of a custom software renderer we use the standard pipeline of OpenGL in order to render the object's silhouettes along

with the depth-buffer. Using hardware rendering, runtime scales well even for complex models composed of a large number of polygons.

To generate synthetic views that match the real images, the camera's intrinsic parameters need to be included (see Eq. 5). We model perspective projection to the canonical view volume v as $\mathbf{X}_v = P_{gl} T_{glc} T^i_{cm} \tilde{\mathbf{X}}^i_m$, with

$$P_{gl} = \begin{bmatrix} \frac{2f_x}{w} & 0 & 1 - \frac{2c_x}{w} & 0 \\ 0 & -\frac{2f_y}{h} & 1 - \frac{2c_y}{h} & 0 \\ 0 & 0 & \frac{Z_f + Z_n}{Z_n - Z_f} & \frac{2Z_f Z_n}{Z_n - Z_f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{and} \quad T_{glc} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{16}$$

where w, h are the camera image width and height, and Z_n, Z_f are the near- and far-plane of the OpenGL view frustum. The matrix T_{glc} describes the transform from the coordinate frame of the real camera to that of the OpenGL camera.

Hierarchical rendering is achieved by changing the width and height of the viewport according to the current pyramid level. As we perform the image processing on the CPU, offscreen rendering to a *FrameBufferObject* is used, which is then downloaded to host memory as the silhouette mask $I_s : \Omega \to \mathbb{R}$ and the corresponding depth-buffer $I_d : \Omega \to \mathbb{R}$. Based on the depth-buffer I_d the surface points $\mathbf{X}^i_c$ can be determined via backprojection as

$$\mathbf{X}^i_c = D(\mathbf{x}_c) K^{-1} \tilde{\mathbf{x}}_c, \text{ with } D(\mathbf{x}_c) = \frac{2 Z_n Z_f}{Z_f + Z_n - (2 I_d(\mathbf{x}_c) - 1)(Z_f - Z_n)}, \tag{17}$$

for all $\mathbf{x}_c \in \Omega^i_f$, where $\tilde{\mathbf{x}}_c = (x_c, y_c, 1)^\top$ denotes the homogeneous extension of the image point $\mathbf{x}_c = (x_c, y_c)^\top$.

3.2 Image processing

For every image I_c and each object m_i we compute two maps $I^i_{Pf}, I^i_{Pb} : \Omega \to \mathbb{R}$ mapping each pixel to its foreground and its background posterior probabiliy as

$$I^i_{Pf}(\mathbf{x}_c) = P^i_f\left(I_c(\mathbf{x}_c)\right) \forall \mathbf{x}_c \in \Omega \quad \text{and} \quad I^i_{Pb}(\mathbf{x}_c) = P^i_b\left(I_c(\mathbf{x}_c)\right) \forall \mathbf{x}_c \in \Omega \tag{18}$$

based on the current color histograms. Due to our hierarchical coarse to fine pose optimization strategy we build a three level image pyramid using a scale factor of 2, resulting in corresponding camera matrices $1/4K, 1/2K$ and K. Overall, we perform six optimization iterations for each object per image. The first three iterations are conducted at the lowest pyramid level, followed by two iterations in the next level and a single iteration at full image resolution.

To distinguish multiple objects, we render each model silhouette region Ω^i_f using a unique intensity corresponding to the model index i. Analog to PWP3D we only evaluate the energy in a narrow band (± 8 pixels) around the contour of each object. Thereby, we use the same width for each pyramid level. All subsequent image processing steps are restricted to a 2D ROI (region of interest) containing this contour band. This ROI is given by the 2D bounding rectangle of the projected bounding box of a model expanded by 8 pixels in each direction.

Fig. 4. Multiple object tracking with occlusion. Left: Input frame showing a red squirrel figurine occluding a green letter *V*. Middle: Silhouette mask I_s of corresponding surface models m_1 and m_2. Right: Φ^2 within ± 8 pixels around the occluded contour C^2 (grey values) and pixels influenced by occlusion (bright red inside, dark red outside of Ω_f^2).

The last three image processing steps are implemented using one thread per CPU core, each processing a bunch of image rows. First, the signed distance transform (Eq. 9) of each silhouette region Ω_f^i is computed. Thereby, we do not explicitly extract the contour using Scharr kernels as done in PWP3D, but use forward finite differences within the algorithm presented in [19]. Next, the derivatives of the signed distance transform are calculated (Eq. 13). Finally, the Hessian approximation and the gradient of the energy needed for the parameter step (Eq. 14) are calculated, where each thread calculates the sums over $(J^i)^\top J^i$ and $(J^i)^\top$ for its pixels, which are added up afterwards in the main thread. Note that only the upper triangular part of $(J^i)^\top J^i$ has to be calculated as it is symmetrical.

3.3 Occlusion Handling

Tracking multiple objects simultaneously mutual occlusions are very likely to emerge (Figure 4), which must be handled appropriately. In our approach occlusions can be modeled using the silhouette mask I_s. Thereby, the natural approach as realized in PWP3D is to render each model's silhouette I_s^i separately and determine its individual signed distance transform Φ^i. Then, the common silhouette mask I_s is also rendered, identifying whether a pixel belongs to the region of a different object and thus has to be discarded for the current object.

Although this approach is easy to compute it does not scale well with the number of objects as each I_s^i, I_d^i and I_s have to be rendered and transferred to host memory. Thereby, computing Φ^i directly from I_d^i avoids downloading I_s^i, but the number of renderings is still linear in the number of objects. In order to minimize rendering and memory transfer we instead render the entire scene once per iteration, download a common silhouette mask I_s and the according depth-buffer I_d, and compute each Φ^i directly from I_s as described in Section 3.2. Thereby, the respective contours C^i can contain segments resulting from occlusions that are considered in the respective signed distance transform. To handle this, for each object all pixels with a distance value that was influenced by occlusion have to be discarded for pose optimization (Figure 4). By only using I_s

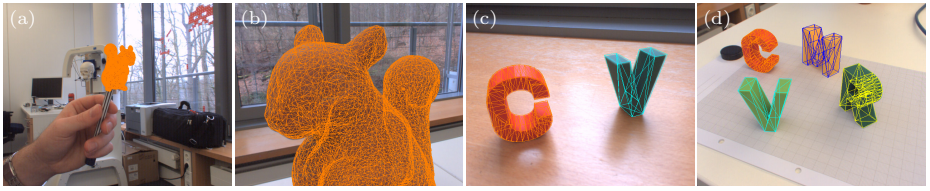

Fig. 5. Example scenes used for performance analysis. The scenes vary in the number of objects, their polygon count and distances to the camera

and I_d this detection is split into two cases. For a pixel $\mathbf{x}_c \in \Omega_b^i$ outside of the silhouette region, we simply check whether $I_s(\mathbf{x}_c)$ equals another object index. In that case, $\mathbf{x}_c$ is discarded if the depth at $I_d(\mathbf{x}_c)$ of the other object is smaller than that of the closest contour pixel to $\mathbf{x}_c$, meaning that the other surface is actually in front of the current object (indicated with dark red in Figure 4). For $\mathbf{x}_c \in \Omega_f^i$ inside of the silhouette region we perform the same checks for all neighboring pixels outside of Ω_f^i to the closest contour pixel to $\mathbf{x}_c$. If any of these pixels next to the contour passes the mask and depth checks, $\mathbf{x}_c$ is discarded (indicated with bright red in Figure 4).

4 Evaluation

This section provides an in-depth performance analysis of our method. We give detailed runtime measurements and experimental evaluations comparing our approach to PWP3D in challenging motion sequences. For all experiments a commodity laptop with Intel Core i7 quad core CPU @ 2.6 GHz and NVIDIA GeForce GT 650M GPU as well as a 640×512 pixel resolution camera were used.

4.1 Performance Analysis

For model-based methods, the runtime not only depends on the number of objects but also on the models' complexity (polygon count and distance to camera), which hamper general performance measurements. We therefore demonstrate the performance of our implementation by providing chronometries (Table 1) of several different example scenes (Figure 5). Note that the significant runtime difference in the third row of Table 1 is due to the fact that we do not need to download the silhouette mask I_s in case of a single object as in this case we can compute Φ^1 directly from I_d.

4.2 Experimental Comparison

We compare our method to PWP3D in three real image data experiments using image sequences pre-recorded at 50 Hz (see https://youtu.be/-nFkNPqf1LU). Our approach was able to track the single object in the first two sequences in

Table 1. Average runtimes (in ms) for the example scenes depicted in Figure 5 for each main processing step combined for all objects and separated by the three pyramid levels. Also the overall runtime per frame is given

Processing step	Figure 5(a)	Figure 5(b)	Figure 5(c)	Figure 5(d)
Posterior pyramid	1.0	1.2	2.5	5.1
Scene rendering	0.2, 0.2, 0.3	0.5, 0.4, 0.4	0.2, 0.2, 0.3	0.3, 0.3, 0.3
Transfer I_d (and I_s)	0.6, 0.4, 0.4	0.6, 0.4, 0.4	1.1, 0.8, 0.7	1.1, 0.8, 0.7
Level-set transforms Φ^i	0.4, 0.2, 0.1	2.9, 0.9, 0.3	1.1, 0.4, 0.3	1.2, 0.7, 0.4
Derivatives $\partial\Phi^i/\partial\mathbf{x}_c$	0.1, 0.05, 0.04	0.2, 0.1, 0.1	0.2, 0.1, 0.1	0.3, 0.2, 0.1
$(J^i)^\top J^i$ and $(J^i)^\top$	0.4, 0.2, 0.1	1.0, 0.5, 0.3	0.9, 0.5, 0.3	1.3, 0.8, 0.6
Histogram update	2.7	3.2	6.4	9.0
Overall per frame	10.4	18.8	22.0	30.2

real-time while PWP3D needed about 60 – 80 ms per frame. In order to neglect the influence of the runtime on the frame rate and the resulting pose difference between consecutive frames, all sequences were post-processed at full frame rate by both algorithms. For PWP3D we used eight optimization iterations since this would enable at least 20 – 25 Hz when using more powerful hardware. While our method was used with the same settings (number of iterations per hierarchy, see Section 3.2) throughout, we had to adjust all step sizes individually for PWP3D for each experiment and each object in order to achieve the best results.

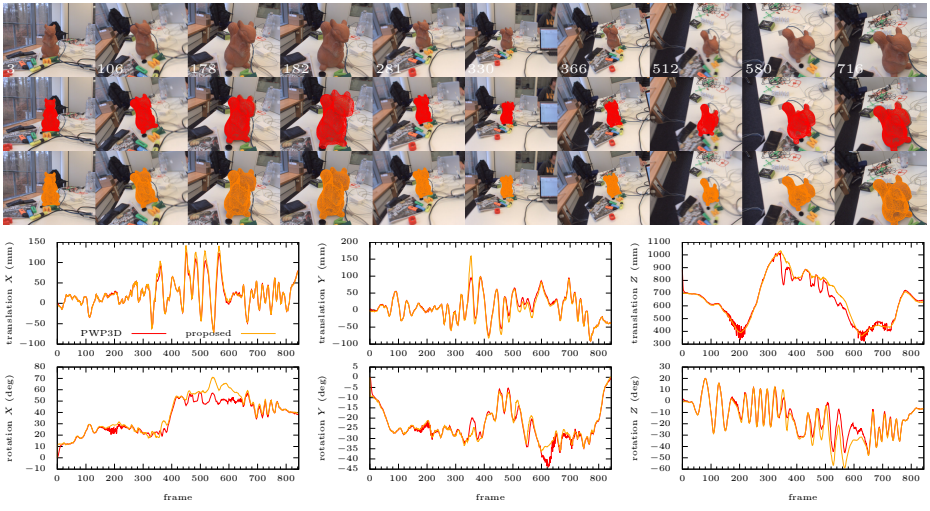

Fig. 6. Top: Visual result of the first pose tracking experiment for both methods (Top row: input images. Middle row: Result of PWP3D. Bottom row: Result of proposed method). Bottom: Determined pose parameters. The color of the plots corresponds to the mesh color drawn in the example frames with respect to the algorithm used

In the first experiment we moved the camera around a scene tracking a stationary squirrel figurine (Figure 6). Thereby, we endeavored to generate fast rotational motion at different distances between camera and object. The results show the dependency of the step sizes in PWP3D on the distance to the camera. If the distance of the object in Z direction becomes too small, the step sizes are too large and the pose starts to oscillate (e.g. frames 150–230), but if the distance between the object and the camera increases the overall optimization quality degrades, resulting in the step sizes to be too small to reach the minimum (e.g. starting around frame 280). We tried to scale the step sizes with regard to the translation in Z-direction which slightly reduced these effects but did not solve the problem. In contrast, our method performed equally well producing pleasing results throughout the whole sequence. If the step sizes of PWP3D suit the distance, both methods perform equally well, which shows that our approach does not degrade the tracking quality (e.g. frames 20–150). At initialization our method always converged within the first three frames (18 iterations) while PWP3D needed at least about 12 frames (96 iterations) until alignment.

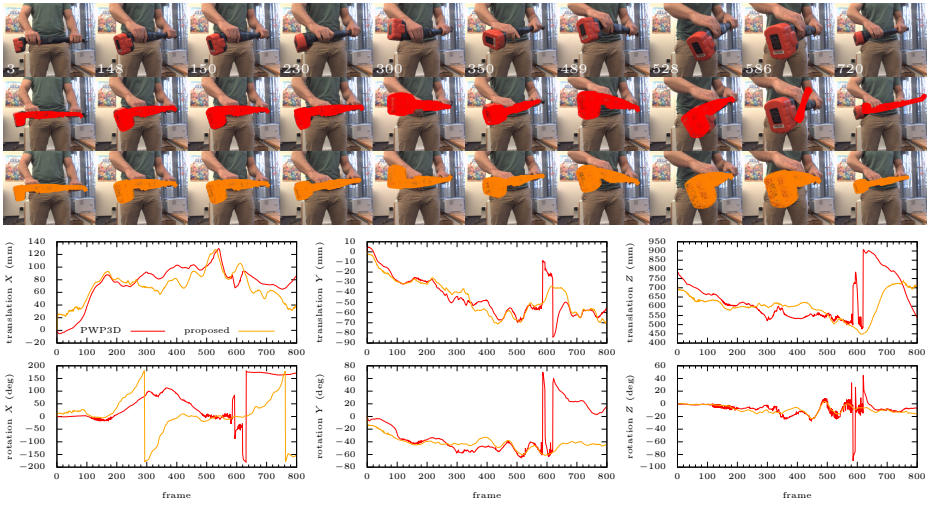

Fig. 7. Results of the second experiment, where a screwdriver was moved in front of the stationary camera. The visualization of the tracking results is analogous to Figure 6

In the second experiment we tracked a hand held screwdriver that was moved in front of the stationary camera (Figure 7). The sequence contains a challenging full 360° turn around the X-axis of the screwdriver that shows the advantage of our parametrization compared to that used in PWP3D (frames 180–400). We set the rotation step size for PWP3D to a large value such that it was close to oscillating (e.g. frames 105–150) since this produced the best overall results. While our method is able to correctly track this motion, PWP3D fails to determine the rotation of the object despite the large step size for rotation. Starting at around

frame 450, the screwdriver was moved closer towards the camera, leading to a tracking loss of PWP3D at frame 586, while the proposed method remained stable even though both methods suffer from poor segmentation caused by the occlusion by the hands and the heterogenous appearance of the screwdriver.

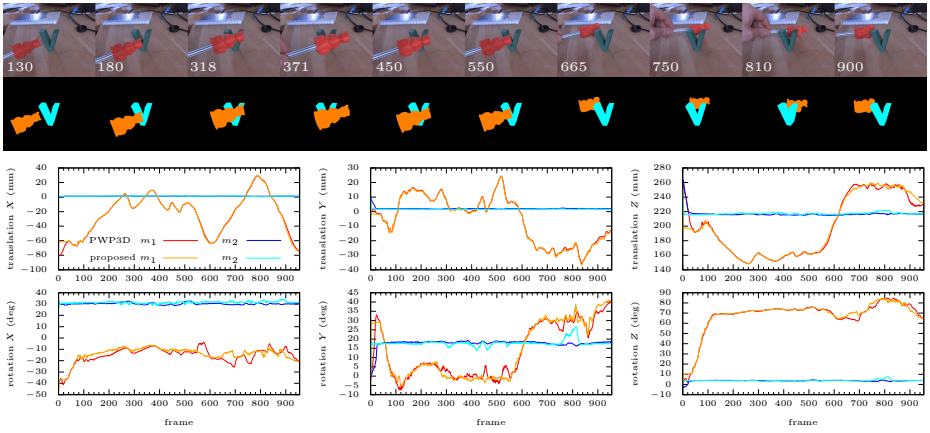

Fig. 8. Results of the third experiment, with two objects occluding one another. Top: Example frames with the proposed corresponding silhouette masks I_s below. Bottom: Plots of determined pose parameters for PWP3D and the proposed method

The third experiment compares our occlusion detection method to the straightforward approach of PWP3D (Figure 8). Here, the green letter V and the camera were stationary while the red squirrel figurine was attached to a stick and moved around. The plots show that both methods struggle with heavy occlusions (e.g. example frames 318, 371, 450) but in general are robust. The unwanted motion of the letter (frames 762–827) was caused by ambient occlusion polluting the color histograms. Due to its overall faster convergence our method in general is more sensitive to these kinds of distortions.

5 Conclusions

We presented an improvement to the current state of the art region-based pose estimation methods. The proposed optimization strategy has fewer parameters to set and better convergence properties compared to previous approaches, which lead to a major increase of tracking robustness and reduction of runtime.

Pose ambiguities caused by object symmetries are inherent to pose estimation methods solely based on silhouette information. In future work we plan on extending the energy functional by a photometric term that incorporates the inner structure such as texture or edges of the objects for resolving these ambiguities. We will also investigate integrating recent approaches improving the image segmentation part (e.g. [16]). Finally detecting tracking losses and subsequent re-localization in real-time were not addressed and remain future work.

References

1. Lepetit, V., Fua, P.: Monocular model-based 3d tracking of rigid objects: A survey. Found. Trends. Comput. Graph. Vis. **1**(1) (January 2005) 1–89
2. Cremers, D., Rousson, M., Deriche, R.: A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape. Int. J. Comput. Vision **72**(2) (April 2007) 195–215
3. Harris, C., Stennet, C.: RAPiD – A video-rate object tracker. In: British Machine Vision Conference. (September 1990) 73–77
4. Vacchetti, L., Lepetit, V., Fua, P.: Stable real-time 3d tracking using online and offline information. IEEE Trans. Pattern Anal. Mach. Intell. **26**(10) (October 2004) 1385–1391
5. Park, Y., Lepetit, V., Woo, W.: Multiple 3d object tracking for augmented reality. In: Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on. (Sept 2008) 117–120
6. Kim, K., Lepetit, V., Woo, W.: Keyframe-based modeling and tracking of multiple 3d objects. In: Mixed and Augmented Reality, 2010. ISMAR 2010. 9th IEEE International Symposium on. (Oct 2010) 193–198
7. Rosenhahn, B., Brox, T., Weickert, J.: Three-dimensional shape knowledge for joint image segmentation and pose tracking. International Journal of Computer Vision **73**(3) (2006) 243–262
8. Rosenhahn, B., Brox, T., Cremers, D., Seidel, H.P.: A comparison of shape matching methods for contour based pose estimation. In Reulke, R., Eckardt, U., Flach, B., Knauer, U., Polthier, K., eds.: Combinatorial Image Analysis. Volume 4040 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 263–276
9. Schmaltz, C., Rosenhahn, B., Brox, T., Cremers, D., Weickert, J., Wietzke, L., Sommer, G. In: Region-Based Pose Tracking. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 56–63
10. Brox, T., Rosenhahn, B., Gall, J., Cremers, D.: Combined region and motion-based 3d tracking of rigid and articulated objects. IEEE Trans. Pattern Anal. Mach. Intell. **32**(3) (March 2010) 402–415
11. Schmaltz, C., Rosenhahn, B., Brox, T., Weickert, J.: Region-based pose tracking with occlusions using 3d models. Machine Vision and Applications **23**(3) (2011) 557–577
12. Prisacariu, V.A., Reid, I.D.: Pwp3d: Real-time segmentation and tracking of 3d objects. Int. J. Comput. Vision **98**(3) (July 2012) 335–354
13. Dambreville, S., Sandhu, R., Yezzi, A., Tannenbaum, A.: A geometric approach to joint 2d region-based segmentation and 3d pose estimation using a 3d shape prior. SIAM J. Img. Sci. **3**(1) (March 2010) 110–132
14. Prisacariu, V., Kahler, O., Murray, D., Reid, I.: Real-time 3d tracking and reconstruction on mobile phones. IEEE Transactions on Visualization and Computer Graphics **21**(5) (May 2015) 557–570
15. Zhao, S., Wang, L., Sui, W., yu Wu, H., Pan, C.: 3d object tracking via boundary constrained region-based model. In: Image Processing (ICIP), 2014 IEEE International Conference on. (Oct 2014) 486–490
16. Hexner, J., Hagege, R.R.: 2d-3d pose estimation of heterogeneous objects using a region based approach. International Journal of Computer Vision (2015) 1–18
17. Murray, R.M., Li, Z., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation. 3 edn. Crc Pr Inc (1994)

18. Bibby, C., Reid, I.: Robust real-time visual tracking using pixel-wise posteriors. In: Proceedings of the 10th European Conference on Computer Vision: Part II. ECCV '08, Berlin, Heidelberg, Springer-Verlag (2008) 831–844
19. Felzenszwalb, P.F., Huttenlocher, D.P.: Distance transforms of sampled functions. Theory of Computing **8**(1) (2012) 415–428
20. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing. 3 edn. Cambridge University Press, New York, NY, USA (2007)