Pauwels, K.; Rubio, L.; Ros, E., "Real-time Pose Detection and Tracking of Hundreds of Objects," Circuits and Systems for Video Technology, IEEE Transactions on , vol.PP, no.99, pp.1,1, doi: 10.1109/TCSVT.2015.2430652
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7102713&isnumber=4358651

Real-time Pose Detection and Tracking of Hundreds of Objects

Karl Pauwels, Leonardo Rubio, and Eduardo Ros

***Abstract*—We propose a novel model-based method for tracking the six-degrees-of-freedom (6DOF) pose of a very large number of rigid objects in real-time. By combining dense motion and depth cues with sparse keypoint correspondences, and by feeding back information from the modeled scene to the cue extraction process, the method is both highly accurate and robust to noise and occlusions. A tight integration of the graphical and computational capability of graphics processing units (GPUs) allows the method to simultaneously track hundreds of objects in real-time. We achieve pose updates at framerates around 40 Hz when using 500,000 data samples to track 150 objects using images of resolution 640×480. We introduce a synthetic benchmark dataset with varying objects, background motion, noise and occlusions that enables the evaluation of stereo-vision-based pose estimators in complex scenarios. Using this dataset and a novel evaluation methodology, we show that the proposed method greatly outperforms state-of-the-art methods. Finally, we demonstrate excellent performance on challenging real-world sequences involving multiple objects being manipulated.**

***Index Terms*—model-based object pose estimation, optical flow, stereo, real time, graphics processing unit (GPU), benchmarking.**

I. Introduction

ESTIMATING and tracking the 6DOF (three translation and three rotation) pose of multiple rigid objects is critical for robotic applications involving object grasping and manipulation, and also for inspection tasks, activity interpretation, or even path planning. In many situations, models of the objects of interest can be obtained quickly and with relative ease, either off-line [1], or on-line in an exploratory stage [2], [3].

A. Related Work

Since real-time pose detection and tracking is such an important ability of robotic systems, a wide variety of methods have been proposed in the past. We only provide a brief overview of the major classes of methods here. Many more examples exist for each class. An important distinction exists between pose *detection* and pose *tracking* methods. Unlike pose tracking methods, pose detection methods do not exploit temporal information. Our work mostly focuses on pose tracking although we rely on pose detection to (re-)initialize tracking and thus also consider the interaction between both. Therefore we also include a brief overview of these methods here.

Pose detection approaches can recover the pose from a single image, without requiring an initial estimate. Many such methods rely on sparse keypoints and descriptors to match 2D image features to 3D model points [4], [5]. It has recently been shown how these methods can scale to a very large number of objects [6], [7]. Template methods on the other hand match images to a set of stored templates covering different views of an object [8]. These templates can contain various features and recently also RGB-D-input has been exploited [9]. When complete 3D object models are available, the estimates provided by such methods can also be subsequently refined using standard depth-based iterative closest point (ICP) procedures [10], [11]. Template-based methods are related to learning-based methods for multi-view object class detection, but the latter typically provide only coarse viewpoint estimates [12].

Pose tracking methods refine a pose estimate, usually obtained at the previous time step, based on the measurements obtained at the current time. The most efficient tracking methods to date match expected to observed edges by projecting a 3D wireframe model in the image [13]. Many extensions have been proposed that exploit also texture information [14], [15], optical flow [16] or particle filtering [5], [17], [18] in order to reduce sensitivity to background clutter and noise. Robustness has also been improved by considering multiple hypotheses [19]. A different class of approaches relies on level-set methods to maximize the discrimination between statistical foreground and background appearance models [20]. These methods can include additional cues such as optical flow and sparse keypoints, but at a large computational cost [21]. Recently, also methods that combine both edge- and region-based information have been proposed [22].

Some of these methods also combine tracking with detection to enable automatic initialization or recovery in case of severe occlusions [4], [18], [23], but the multiple object case is not considered explicitly as this requires accounting for inter-object occlusion and having scalable computational requirements. So far, these aspects have been mostly studied in the context of two-dimensional tracking [24] rather than full 6-DOF pose estimation.

Most of the above-mentioned tracking approaches can exploit multi-view information or blob-based stereo triangulation [25], but dense depth information is rarely used. Particle filtering has been used recently for RGB-D object tracking [26] but the dense depth information is only used there to evaluate the particle hypotheses, rather than to compute a pose update, as done in ICP-based methods, such as the one

K. Pauwels is with the Computer Vision and Active Perception lab, Royal Institute of Technology (KTH), Stockholm, Sweden, e-mail: kpauwels@kth.se.

L. Rubio is with Fuel 3D Technologies Limited, Oxford, UK, e-mail: leo@fuel-3d.com.

E. Ros is with the Computer Architecture and Technology Department, University of Granada, Spain, e-mail: eros@ugr.es.

proposed here. An ICP approach [10] can be used provided the object has sufficiently salient shape features. This has been applied with great success in related problems such as online scene modeling [2] where the whole scene is considered as a rigid object. Recently, due to the prevalence of cheap depth sensors, depth information is also being applied in other related problems, such as articulated body pose estimation [27] and visual servoing [28].

We show here that incorporating some of these advances in real-time object tracking, and extending them with additional cues, yields great improvements.

B. Novelty

Unlike most model-based methods that exploit only salient parts of the model (keypoints, silhouettes, edges, shape features), we instead aim to retain all of the model's shape and appearance information, and match it to the observed dense visual features. In this way, the useful feature-set is constrained by the dense algorithms at perception time, rather than at model creation time.

The main contributions can be summarized as follows. Firstly, we introduce a model-based 6DOF pose detection and tracking method that combines dense motion and stereo disparity measurements with sparse keypoint features. It exploits feedback from the modeled scene to the cue extraction level and provides an effective measure of reliability. Although dense motion and stereo have been combined before in model-free 6DOF camera pose estimation [29], we show here how the model information can be used to keep the motion and stereo measurements separate while still jointly minimizing their cost in the error function. For the stereo component we use an ICP approach rather than the 'disparity flow' used by [29]. We show that by iteratively re-rendering the scene, we can move from the differential pose updates typically obtained from linearized edge- or motion-based energy functions, to the full discrete pose update. Secondly, the method has been designed specifically for high-performance operation ($\pm$ 40 Hz with 150 objects). By allowing a tracking system to operate at higher framerates, the motion flows between successive frames become smaller which simplifies tracking and allows for faster object motion [30]. To achieve this, every aspect of the algorithms and the system has been developed with GPU acceleration in mind. Both the graphics and computation pipelines of a modern GPU are extensively used and tightly integrated in both the low-level cue extraction and pose tracking stages. Finally, an extensive benchmark dataset and evaluation methodology have been developed and used to show increased performance (in accuracy, robustness, and speed) as compared to state-of-the-art methods.

Parts of this work have been presented earlier [31]. We introduce here an extension to the multi-object case, together with more method and implementation details, a more detailed evaluation, and additional real-world results.

II. Proposed Method

A concise overview of the method is shown in Fig. 1. Different visual cues are extracted from a stereo video stream and combined with model information (known a priori) to estimate the 6DOF pose of multiple objects. In turn, scene information (related to the joint appearance and shape of the tracked models) is fed back to facilitate the cue extraction itself.

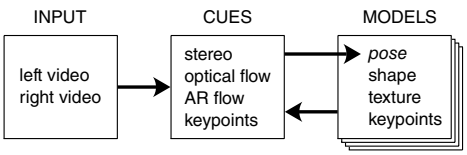

Fig. 1. Method overview illustrating the different visual cues (AR = augmented reality, see II-B2) and model components. The cues are combined to estimate the objects poses and scene information is fed back to facilitate the cue extraction.

A. Scene Representation

The surface geometry and appearance of the objects to track are modeled by a triangle mesh and color texture respectively. Modern graphics hardware allows for these models to be highly complex in shape and appearance. For a given set of model poses, the color (Fig. 2B), distance to the camera (Fig. 2E), surface normal (Fig. 2F), and object identity (Fig. 2D) can be obtained efficiently at each pixel through OpenGL rendering. As a result also self-occlusions and occlusions between different modeled objects are handled automatically through OpenGL's depth buffer. In a training stage, SIFT (Scale-Invariant Feature Transform) features [32] are extracted from keyframes of rotated versions of each model (30° separation), and mapped onto the model's surface.

B. Visual cues

The dense motion and stereo cues are obtained using coarse-to-fine GPU-accelerated phase-based algorithms [33], [34] (modified as discussed next), and the SIFT features are extracted and matched using a GPU library [35].

1) Model-based dense stereo: Coarse-to-fine stereo algorithms, although highly efficient, support only a limited disparity range and have difficulties detecting fine structures [36]. To overcome these problems we feed the object pose estimates obtained in the previous frame back as a prior in the stereo algorithm. Figure 3A,B show an example real-world stereo image pair of a box being manipulated. Using the box's previous frame pose estimate, stereo priors are generated for the current frame pair by converting OpenGL's Z-buffer to disparities for both the left and right cameras. These disparity values are downsampled (Fig. 3C,D) and introduced at the lowest scale used by the stereo algorithm. Stereo disparity is then computed twice, once with respect to the left and once with respect to the right image (essentially by swapping left and right images), each time processing the entire pyramid. This results in two separate disparity maps, one from the left to the right image, δ_L^R, and one from the right to the left image, δ_R^L. A left/right consistency check is then used to remove unreliable estimates. Concretely, δ_L^R is used to find the

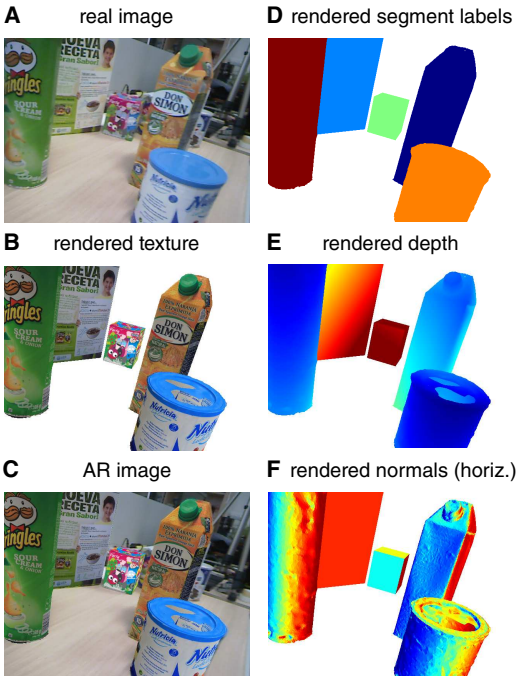

Fig. 2. (A) Original input image, (B) color image obtained by rendering the textured models according to the current pose estimates, (C) AR image obtained by merging A and B (see II-B2), (D) segment (object identity) labels accounting for occlusions, (E) depth-buffer, and (F) horizontal component of the normals. Note the model errors in the lid of the *Nutricia* object.

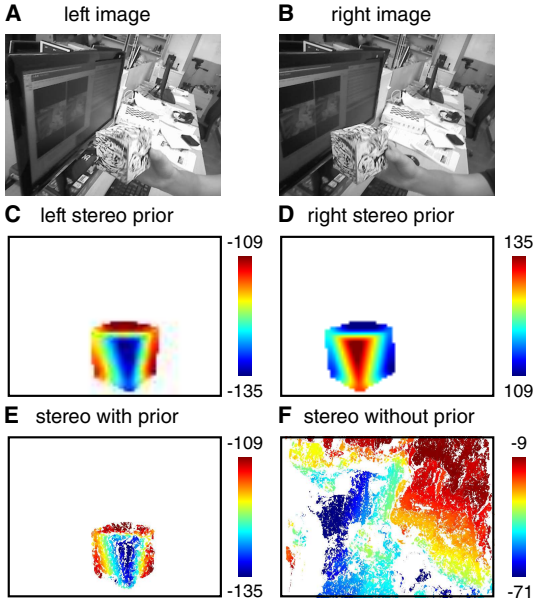

Fig. 3. (A) Left and (B) right input images and low-resolution (C) left and (D) right stereo priors generated using the previous frame pose estimate of the manipulated box. (E) Stereo obtained using four scales and initializing with the prior, and (F) stereo obtained using six scales and without using the prior.

corresponding pixel in δ_R^L and the following error is evaluated in each pixel:

$$e_\delta(\mathbf{x}) = |\delta_L^R(\mathbf{x}) + \delta_R^L\big(\mathbf{x} + \delta_L^R(\mathbf{x})\big)| \ . \tag{1}$$

Disparity estimates with an error exceeding 0.5 pixel are then considered unreliable (see [36] for further details). The reliable estimates for the current frame obtained with and without the priors are shown in panels E and F respectively. For the prior-less algorithm, we used the maximal amount of scales (six) that allow us to fit the filter kernels (11×11) at the lowest resolution. Due to the large range of disparities in this particular scene, without the prior, the focus is on the background rather than on the object of interest (Fig. 3F). Note that the prior corresponds to the *previous* frame pose estimate. With prior, the stereo algorithm thus only has to correct the disparity that results from the pose difference between the current and previous frame. This is much smaller than the absolute disparity and is instead of the order of the correspondences obtained in an optical flow scenario. In Fig. 3E we show that four scales are sufficient to obtain detailed estimates. In certain scenarios (*e.g.* complete loss of tracking) it is possible that a completely wrong prior is generated. In this case the pose selection mechanism (see II-C2) will signal this and either switch to the sparse detector's pose estimate, or mark both the tracker and detector estimates unreliable.

2) Dense motion cues: The optical flow algorithm integrates the temporal phase gradient across different orientations and also uses a coarse-to-fine scheme to increase the dynamic range [37]. Unlike [37], we estimate the temporal phase gradient using two instead of five frames, more specifically the frames I_t and I_{t+1}, at times t and $t+1$. This results in noisier estimates, but reduces the latency. This trade-off can be made depending on the expected dynamics of the considered scenario. In a similar fashion as in the stereo algorithm, a simple consistency check is used to discard unreliable estimates (in this case a forward/backward as opposed to left/right consistency check in the stereo case). A prior is not required here since displacements in the motion scenario are much smaller than in the stereo scenario. The image resolution considered here allows for the use of six scales which enables a maximal optical flow velocity of around 64 pixels/frame. We refer to [33], [36] for more specific details of the algorithms.

Figure 4C contains the (subsampled and scaled) optical flow vectors from Fig. 4A to the next image in a complex real-world scenario with both object and camera motion. Besides the optical flow, we also extract a second type of motion which we refer to as augmented reality (AR) flow that incorporates scene-feedback. The models' textures are rendered at their current pose estimates and overlaid on I_t, resulting in an 'augmented image' $\hat{I}_t$. An example is shown in Fig. 4B using a single object (see Fig. 2C for a multi-object example). The motion is then computed from $\hat{I}_t$ to the next real image (I_{t+1}), and shown in Fig. 4D for the example presented here. Because of the erroneous pose estimate (deliberately large in this case to better illustrate the concept) this motion is quite different from the optical flow. It allows the tracker to recover from such errors by effectively countering the drift that results from tracking based on optical flow alone.

Note that the appearance (especially the brightness) of the model texture will always be somewhat different from the real

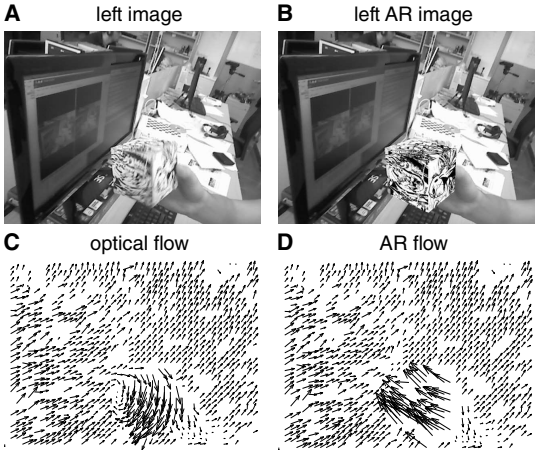

Fig. 4. (A) Left image and (B) left image with rendered model (at incorrect pose) superimposed. (C) Real optical flow from real image in A to the next real image. (D) AR flow from AR image in B to the next real image. Both flow fields are subsampled (15×) and scaled (5×).

scene. It is therefore critical that the optical flow algorithm is largely invariant to this. Since the phase-based algorithm used here relies on a *phase*-constancy rather than a *brightness*-constancy assumption, it is inherently invariant to intensity differences. Consider the more extreme (but not unrealistic) example in Fig. 5. Here, the images were recorded with short exposure, resulting in large intensity differences between the real scene and the model texture. We compare the optical flow of our method to a standard pyramidal Lucas & Kanade algorithm (L&K) [38]. Note that the optical flow (from 5A to 5B) is similar for both methods (although L&K is more noisy). L&K however fails completely on the AR flow (from 5C to 5B) where the large intensity differences result in large flow vectors (5E). The phase-based algorithm is completely insensitive to this and correctly perceives the rotation of the AR image (5G). The same default parameters were used in (D,E) and (F,G).

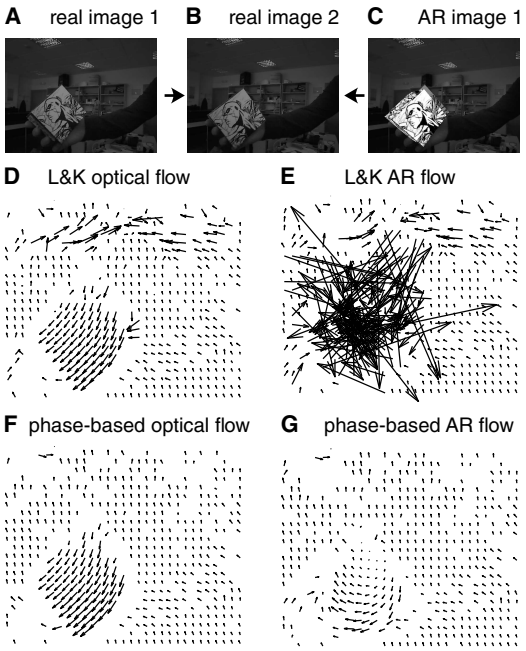

Fig. 5. Importance of intensity invariance for AR flow computation. (A) and (B) are consecutive images obtained with short exposure. The AR image (C) is composed by superimposing the model (with much brighter texture) at an incorrect pose on image (A). Optical flow from A to B computed with (D) pyramidal Lucas & Kanade and (F) our phase-based algorithm. AR flow from C to B with (E) pyramidal Lucas & Kanade and (G) phase-based. All flow fields are subsampled (20×) but unscaled.

C. Pose detection and Tracking

The proposed tracking method incorporates the differential rigid motion constraint into a fast variant of the ICP algorithm [10] to allow all the dense cues to simultaneously and robustly minimize the pose errors. A selection procedure (II-C2) is used to re-initialize the tracker with estimates obtained from a sparse keypoint-based pose detection approach, if required.

1) Dense Pose Tracking: In the following we will consider the single object case only. Multiple objects can be treated independently and in parallel since we are not considering interactions between the objects. Note that occlusions are handled automatically at the rendering stage. In Section III we will discuss in more detail how this parallel optimization is performed.

Our aim is to recover the rigid rotation and translation that best explains the dense visual cues and transforms each model point $\mathbf{m} = [m_x, m_y, m_z]^{\mathrm{T}}$ at time t into point $\mathbf{m}'$ at time $t+1$:

$$\mathbf{m}' = \mathrm{R}\,\mathbf{m} + \mathbf{t} \ , \tag{2}$$

with R the rotation matrix and $\mathbf{t} = [t_x, t_y, t_z]^{\mathrm{T}}$ the translation vector. The rotation matrix can be simplified using a small angle approximation:

$$\mathbf{m}' \approx (\mathbf{1} + [\boldsymbol{\omega}]_\times)\,\mathbf{m} + \mathbf{t} \ , \tag{3}$$

with $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^{\mathrm{T}}$ representing the rotation axis and angle. Each stereo disparity measurement, d', at time $t+1$ can be used to reconstruct an approximation, $\mathbf{s}'$, to $\mathbf{m}'$. Since we are using a rectified camera configuration, this is quite straightforward:

$$\mathbf{s}' = \begin{bmatrix} s'_x \\ s'_y \\ s'_z \end{bmatrix} = \begin{bmatrix} x\,s'_z/f \\ y\,s'_z/f \\ -f\,b/d' \end{bmatrix} \ , \tag{4}$$

with $\mathbf{x} = [x, y]^{\mathrm{T}}$ the pixel coordinates (with nodal point as origin), f the focal length, and b the baseline of the stereo rig. The focal length and baseline are assumed known and are approximately 500 pixels and 70 mm respectively in the stereo system used in the remainder. To use this reconstruction in (2), the model point $\mathbf{m}$ corresponding to $\mathbf{s}'$ needs to be found. We deliberately avoid using the motion cues to facilitate this correspondence search (as done in many ICP extensions) since this requires both valid stereo and valid motion measurements at the same pixel. Instead we use the efficient projective data association algorithm [39] that corresponds stereo measurements to model points that project to the same pixel. These correspondences can be obtained instantly, but they are less accurate. Therefore, a *point-to-plane* as opposed to a *point-to-point* distance needs to be minimized [40], which is obtained

by projecting the error on $\mathbf{n} = [n_x, n_y, n_z]^{\mathrm{T}}$, the model normal vector in $\mathbf{m}$:

$$e = \Big[(\mathrm{R}\,\mathbf{m} + \mathbf{t} - \mathbf{s}') \cdot \mathbf{n}\Big]^2 \ . \tag{5}$$

Note that the normal, obtained by rendering the scene through OpenGL, is already in the camera frame (Fig. 2F). This error expresses the distance from the reconstructed points to the plane tangent to the model. By linearizing this measure as in (3) and summing over all correspondences $\{\mathbf{m}_i, \mathbf{s}'_i\}$, we arrive at a strictly shape-based error measure that is linear in the parameters of interest $\mathbf{t}$ and $\boldsymbol{\omega}$:

$$e_S(\mathbf{t}, \boldsymbol{\omega}) = \sum_i \left(\Big[(\mathbf{1} + [\boldsymbol{\omega}]_\times)\, \mathbf{m}_i + \mathbf{t} - \mathbf{s}'_i \Big] \cdot \mathbf{n}_i \right)^2 \ . \tag{6}$$

We next discuss the incorporation of dense motion estimates into the pose tracking procedure. By rearranging (3), we obtain the following:

$$\mathbf{m}' - \mathbf{m} \approx \mathbf{t} + \boldsymbol{\omega} \times \mathbf{m} \ . \tag{7}$$

Note that this looks very similar to the differential motion equation from classical kinematics that expresses the 3D motion of a point, $\dot{\mathbf{m}}$, in terms of its 3D translational and rotational velocity:

$$\dot{\mathbf{m}} = \mathbf{t} + \boldsymbol{\omega} \times \mathbf{m} \ . \tag{8}$$

We deliberately retain the $(\mathbf{t}, \boldsymbol{\omega})$ notation since the displacements in (3) are expressed in the same time unit. This can now be used with the optical flow to provide additional constraints on the rigid motion. Unlike in the stereo case (4), $\dot{\mathbf{m}}$ cannot be reconstructed from the optical flow and instead (8) needs to be enforced in the image domain. After projecting the model shape:

$$\mathbf{x} = f \begin{bmatrix} m_x/m_z \\ m_y/m_z \end{bmatrix} \ , \tag{9}$$

the expected pixel motion $\dot{\mathbf{x}} = [\dot{x}, \dot{y}]^{\mathrm{T}}$ becomes:

$$\dot{\mathbf{x}} = \frac{\delta \mathbf{x}}{\delta t} = \frac{f}{m_z^2} \begin{bmatrix} \dot{m}_x m_z - m_x \dot{m}_z \\ \dot{m}_y m_z - m_y \dot{m}_z \end{bmatrix} \ . \tag{10}$$

Combining (10) with (8) results in the familiar equations [41]:

$$\dot{x} = \frac{(f\,t_x - x\,t_z)}{m_z} - \frac{x\,y}{f}\,\omega_x + (f + \frac{x^2}{f})\,\omega_y - y\,\omega_z \ , \tag{11}$$

$$\dot{y} = \frac{(f\,t_y - y\,t_z)}{m_z} - (f + \frac{y^2}{f})\,\omega_x + \frac{x\,y}{f}\,\omega_y + x\,\omega_z \ , \tag{12}$$

which are linear in $\mathbf{t}$ and $\boldsymbol{\omega}$ provided the depth of the point is known. We obtain this depth m_z by rendering the model at the current pose estimate rather than using the stereo measurement (9). This has the advantage of keeping the motion and stereo measurements strictly separate. Since we have two sources of pixel motion, we have two error functions:

$$e_O(\mathbf{t}, \boldsymbol{\omega}) \ = \ \sum_i \|\dot{\mathbf{x}}_i - \mathbf{o}_i\|^2 \ , \tag{13}$$

$$e_A(\mathbf{t}, \boldsymbol{\omega}) \ = \ \sum_i \|\dot{\mathbf{x}}_i - \mathbf{a}_i\|^2 \ , \tag{14}$$

with $\mathbf{o} = [o_x, o_y]^{\mathrm{T}}$ and $\mathbf{a} = [a_x, a_y]^{\mathrm{T}}$ the observed optical and AR flow respectively.

Both the linearized point-to-plane distance in the stereo case and the differential motion constraint in the optical and AR flow case now provide linear constraints on the same rigid motion representation $(\mathbf{t}, \boldsymbol{\omega})$ and can thus be minimized jointly. Note that the optical flow errors are expressed in pixels, whereas the shape-based error is measured in Euclidian space. This is problematic since the latter does not correctly weigh each sample according to the expected noise. The noise distributions of the disparity and optical flow measurements can be expected to be similar in image space, since both are obtained using similar phase-based correspondence finding algorithms. We can resolve the difference in units by using a weighted version of (6) instead:

$$e_{SW}(\mathbf{t}, \boldsymbol{\omega}) = \sum_i \left(w_i \Big[(\mathbf{1} + [\boldsymbol{\omega}]_\times)\, \mathbf{m}_i + \mathbf{t} - \mathbf{s}'_i \Big] \cdot \mathbf{n}_i \right)^2 \ , \tag{15}$$

where we use the derivative of the depth-to-disparity transform, $d(z) = -fb/z$, see (4), evaluated at the sample's model depth m_{z_i}:

$$w_i = \frac{fb}{m_{z_i}^2} \ . \tag{16}$$

This weight effectively transforms a change in depth, as measured in (6), into a change in disparity. For stability, we use the model depth at the current pose rather than the depth estimated from disparity. Now that each component of the error is expressed in the image domain, we simply combine them to arrive at the following error function:

$$E(\mathbf{t}, \boldsymbol{\omega}) = e_{SW}(\mathbf{t}, \boldsymbol{\omega}) + e_O(\mathbf{t}, \boldsymbol{\omega}) + e_A(\mathbf{t}, \boldsymbol{\omega}) \ . \tag{17}$$

We let each cue contribute equally in this error function. The optimal balance between optical and AR flow depends on model quality and is therefore hard to decide in advance. Since the system as a whole is quite robust due to the multiple cues, this issue is not critical. More important though is the relative weighting of stereo and motion. For this purpose we evaluated different weight settings on the entire benchmarking dataset of Section IV and observed only marginal effects in a range around equal weight. Setting weights to zero does have large effects though, as will be shown in Section V-A.

To increase robustness, an M-estimation scheme is used to gradually reduce the influence of outliers and ultimately remove them from the estimation [42]. In the case of large rotations the linearized constraints used in (17) are only crude approximations, and many of the shape correspondences obtained through projective data association will be wrong. Therefore, the minimization of (17) needs to be iterated a number of times, at each iteration updating the pose, the shape correspondences, and the unexplained part of the optical and AR flow measurements.

At iteration k, an incremental pose update is obtained by minimizing $E(\Delta\mathbf{t}^k, \Delta\boldsymbol{\omega}^k)$, and accumulated into the pose estimate at the previous iteration $k-1$:

$$\mathrm{R}^k \ = \ \Delta\mathrm{R}^k\,\mathrm{R}^{k-1} \ , \tag{18}$$

$$\mathbf{t}^k \ = \ \Delta\mathrm{R}^k\,\mathbf{t}^{k-1} + \Delta\mathbf{t}^k \ , \tag{19}$$

where $\Delta\mathrm{R}^k = e^{[\Delta\boldsymbol{\omega}^k]_\times}$. The model is updated as:

$$\mathbf{m}^k = \mathrm{R}^k\,\mathbf{m} + \mathbf{t}^k \ , \tag{20}$$

and used to obtain the new (projective) shape correspondences and the part of the optical and AR flow explained thus far, $\Delta\mathbf{x}^k = [\Delta x^k, \Delta y^k]^{\mathrm{T}}$:

$$\Delta x^k = f(m_x^k/m_z^k - m_x/m_z) \ , \quad (21)$$
$$\Delta y^k = f(m_y^k/m_z^k - m_y/m_z) \ . \quad (22)$$

This explained flow is subtracted from the observed optical and AR flow:

$$\mathbf{o}^k = \mathbf{o} - \Delta\mathbf{x}^k \ , \quad (23)$$
$$\mathbf{a}^k = \mathbf{a} - \Delta\mathbf{x}^k \ . \quad (24)$$

The next iteration incremental pose updates are then obtained by minimizing $E(\Delta\mathbf{t}^{k+1}, \Delta\boldsymbol{\omega}^{k+1})$, which operates on $\mathbf{o}^k$, $\mathbf{a}^k$, $\mathbf{m}^k$, and $\mathbf{s}'$. This cycle is repeated a fixed number of times (we use three internal iterations for the M-estimation, and three external iterations). Note that even though the updates in each iteration are estimated from linearized equations, the correct accumulated discrete updates are used in between iterations (20–22).

2) Combined Sparse and Dense Pose Estimation: A RANSAC-based monocular perspective-n-point pose estimator is used to robustly extract the 6DOF object pose on the basis of correspondences between image (2D) and model codebook (3D) SIFT keypoint descriptors [38], [43]. Exhaustive matching is used and therefore, unlike the dense tracking component of the proposed method, this sparse estimator provides a pose estimate that does not depend on the previous frame's estimate.

Due to the nonlinear and multimodal nature of the sparse and dense components, directly merging them using for instance a Kalman filtering framework is not suitable here. Instead we *select* either the sparse or dense estimate based on the effectiveness of its feedback on cue extraction, as measured by the proportion of valid AR flow vectors in the projected (visible, unoccluded) object region. An example of these regions is shown in Fig. 2D. The pose estimate that leads to the largest proportion valid AR flow wins. As mentioned above, a flow vector is considered valid if it passes a simple forward/backward consistency check (II-B2). Note that the accuracy of AR flow is affected by model inaccuracies, but since we are comparing (dense-pose-based) AR flow to (sparse-pose-based) AR flow, both will be equally affected. A very important characteristic of this measure is that, unlike optical-flow-based or stereo-based measures, AR flow *is* affected by occlusions (unless the occlusion is due to another tracked object). When the occluder becomes dominant, the dense tracker will start using the occluder's motion and stereo measurements for the pose update, rather than those of the object-of-interest. At this point the sparse pose update will be selected since its more accurate pose estimate will likely result in a larger AR flow density. This will remain active until the occlusion becomes too large. Then, the small proportion of valid AR flow will signal the problem and the object will be considered lost (occluded). It then needs to be re-detected for tracking to resume. We show in Section V that this simple measure is adequate for selecting and determining the reliability of the pose estimate.

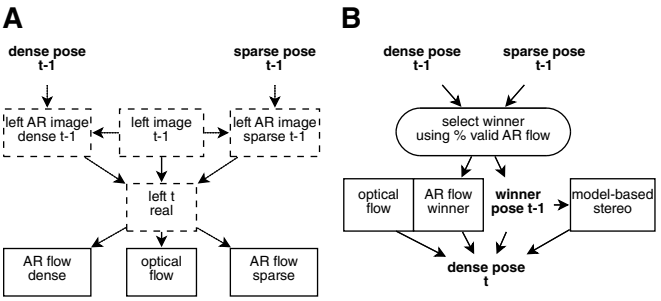

Fig. 6. Combined sparse pose detection and dense pose tracking. Dashed boxes refer to images and solid boxes to extracted cues. (A) Processing sequence for obtaining the three different motion cues. (B) Sequence performed to obtain the dense cues and select the winner pose. Note that model-based stereo only needs to be computed once since it is not used in the evaluation.

Figure 6A shows how the previous frame dense and sparse poses are used to generate two AR images. From these, the dense and sparse AR flow are extracted and the winner pose is selected based on the proportion of valid AR flow vectors in the object region (Fig. 6B). This pose is then used to update the stereo priors and obtain the model-based stereo at time t. Finally, all cues are combined in the tracker resulting in the dense pose at time t.

3) Multiple Object Pose Estimation: The above-mentioned approach can be extended to the multi-object case as follows. Since the sparse estimator does not scale to multiple objects like the tracker, we let it operate on a single (but possibly different) object at each time instance. This object is selected probabilistically according to the current reliability of each object's pose. Concretely, each object o has the following probability of being selected for a sparse update:

$$p(o) = \frac{1 - r(o)}{\sum_i \big(1 - r(i)\big)} \ , \quad (25)$$

with $r(o)$ the reliability (*i.e.* the proportion valid AR flow). This functions as an attention mechanism, focusing the limited resources on the least reliable estimates. With multiple objects present, it is critical to consider the reliability of the object poses at rendering time. Rendering an unreliable object at an incorrect pose (for example close to the camera) can dramatically disturb the processing of the other objects due to the occlusion handling in the Z-buffering. Therefore, unreliable objects are not considered in the tracking stage. They instead receive a higher probability of becoming the sparse pose estimator's current target.

The sparse/dense selection is performed in a similar way as in the single object scenario (considering the reliability of the updated object) but now also ensuring that the sparse update does not negatively affect the reliability of any other currently tracked object.

III. GPU Implementation

An overview of all the processing steps in the multi-object case is provided in Algorithm 1. The entire system has been developed using OpenGL and NVIDIA's CUDA framework [44].

Algorithm 1: GPU multiple pose tracking update

```
input : initial poses; low-level visual cues
        (Section III-A)
output: pose updates
for ICP iterations do
    begin pre-process (Section III-B)
        mark valid samples with OpenGL segment labels
        (Fig. 2D)
        radix sort indices of valid samples based on
        segment labels
        if #samples > max_samples then
            subsample valid indices
        gather residual optical and AR flow, disparity,
        and OpenGL depth and normals
    end
    begin ordinary least squares (Section III-C)
        compose normal equations optical and AR flow
        compose normal equations disparity
        solve systems (on CPU)
    end
    for reweighting iterations do
        begin robust least squares iteration
        (Section III-D)
            compute abs. residuals optical and AR flow
            compute abs. residuals disparity
            compute approx. median abs. residuals
            compose weighted normal equations optical
            and AR flow
            compose weighted normal equations disparity
            solve systems (on CPU)
        end
    compute residual optical and AR flow (23,24)
    update poses
    render scene at updated poses
```

A. Low-level Vision

As shown in Fig. 6 the AR flow is computed twice, based on the previous frame's sparse and dense pose estimates. Consequently, four Gabor pyramids need to be computed (left image, right image, sparse left AR image, dense left AR image). A number of rendering steps are also required to create the stereo priors and AR images. A detailed discussion of the GPU implementation of the low-level component of the system (but without the feedback components introduced here) can be found in [33].

B. Pre-processing

The pre-processing component aims to re-organize the pixel-based low-level vision and model-based cues to enable efficient processing in subsequent stages. Concretely this involves assigning all valid measurements (residual optical and AR flow, disparity, Z-buffer and normals) to their respective segments, in accordance with the current pose estimates. A unique index or label is associated with each object and by rendering the scene according to the current pose estimates, the corresponding segment label (or a label signaling no object) is assigned to each pixel (Fig. 2D). All OpenGL data written by the shaders is directly accessible through CUDA. The most expensive operation at this stage is sorting the label indices. Since every image pixel has a label, a general sorting operation is infeasible. However, the number of labels is limited and an efficient radix sort can be used instead [45]. After sorting, the indices are subsampled if the total available measurements exceeds a threshold ($max\_samples$). All data is then gathered and a compacted stream results.

C. Ordinary Least Squares

Least squares estimation involves composing and solving the normal equations corresponding to the linear least squares system of (17). We can rewrite this as follows to expose the linearity:

$$\begin{aligned} E(\boldsymbol{\alpha}) \quad = \quad & (\mathbf{F}_{SW}\boldsymbol{\alpha} - \mathbf{d}_{SW})^{\mathrm{T}}(\mathbf{F}_{SW}\boldsymbol{\alpha} - \mathbf{d}_{SW}) + \\ & (\mathbf{F}_{M}\boldsymbol{\alpha} - \mathbf{d}_{M})^{\mathrm{T}}(\mathbf{F}_{M}\boldsymbol{\alpha} - \mathbf{d}_{M}) \ , \end{aligned} \tag{26}$$

where the rigid motion parameters are stacked into a screw vector $\boldsymbol{\alpha} = \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{t} \end{pmatrix}$ for convenience, and $\mathbf{F}_{SW}$, $\mathbf{d}_{SW}$ and $\mathbf{F}_{M}$,$\mathbf{d}_{M}$ are obtained by gathering the sensor data according to (15) and (11,12) respectively (for compactness, we no longer distinguish between optical and AR flow at this point). This can be solved in the least-squares sense using the normal equations:

$$(\mathbf{F}_{SW}^{\mathrm{T}}\mathbf{F}_{SW} + \mathbf{F}_{M}^{\mathrm{T}}\mathbf{F}_{M})\boldsymbol{\alpha} = (\mathbf{F}_{SW}^{\mathrm{T}}\mathbf{d}_{SW} + \mathbf{F}_{M}^{\mathrm{T}}\mathbf{d}_{M}) \ . \tag{27}$$

The construction of the matrices in the lefthand-side of (27) involves a substantial increase of data at each sample. A sample here refers to a valid optical or AR flow vector, or a valid stereo disparity measurement. For example, for the motion case (11,12) each sample contains the motion vector (2D), the depth-buffer (1D), and the (linearized) pixel location (1D), resulting in a total of 4 input values. The flow normal equations associated with this sample however contain 23 unique values (due to symmetry etc.). For the stereo case (15) the expansion goes from 6 input values (1 disparity, 1 depth, 3 normals, 1 pixel location) to 27 unique values in the normal equations. To limit the data stream, this composition is combined with an initial compaction operation. This is then followed by an additional GPU kernel (a CUDA processing step) to completely reduce the normal equations. These normal equations are then solved on the CPU.

D. Reweighted Least Squares

Tukey's M-estimation scheme [42] (an iteratively reweighted least squares approach) is used to robustly solve the normal equations in the presence of outliers. At each iteration the least squares problem from the previous section is solved, but now weighting each constraint inversely proportional to the error obtained with the current estimate. Samples whose error exceeds a certain threshold are completely removed from the estimation.

Critical here is the scale of the absolute residual distribution, which is directly related to this outlier rejection threshold. This involves computing the median for each segment (object)

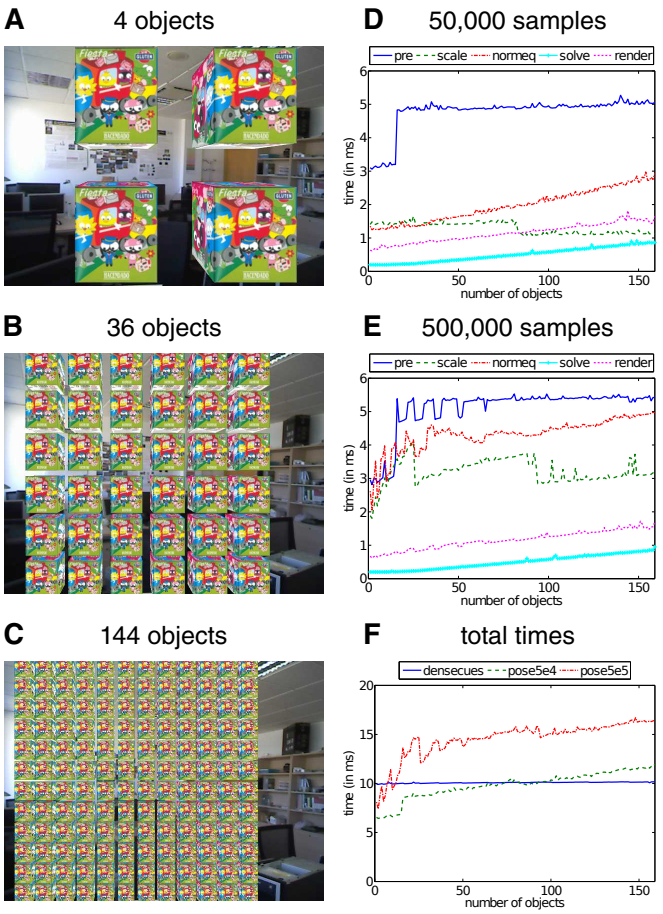

Fig. 7. (A,B,C) Example synthetic images used to evaluate computational performance as a function of number of objects tracked. Computation times of the pose update component for (D) 50,000 and (E) 500,000 samples. The legend refers to the following stages: pre-processing *(pre)*, absolute residuals and scale *(scale)*, composition and reduction of the normal equations *(normeq)*, solving the normal equations *(solve)* and rendering *(render)*. The total computation times for the dense cue extraction *(densecues)* and pose estimation using 50,000 *(pose5e4)* and 500,000 *(pose5e5)* samples are shown in (F). These times are lower than the total times in (D) and (E) due to the reduced overhead of timing the components.

at each iteration. Since computing the exact median requires too many sorting operations, we instead use an approximation to the median, as successfully used previously in [46]. The particular algorithm used performs a recursive reduction operation on triplets, at each stage replacing each triplet by its center value [47]. This algorithm is very suitable for GPU implementation. We limit the maximum number of elements for determining this approximation to $3^9 = 19,683$ per segment. We did not find any significant reduction in accuracy when using the approximate rather than exact median on the entire benchmarking dataset of Section IV. Concretely, the absolute residuals are computed, the scale is determined, and then the normal equations are computed as in the previous section, but now weighting each sample according to the scale and the residual. The final systems are again solved on the CPU.

E. Processing Times

We have created a synthetic problem dataset to evaluate how the computation times scale as a function of the number of objects being tracked. In this experiment the images were of resolution 640×480. The same object was rendered multiple times in a regular grid. The left camera images for the first frame for problems involving 4, 36 and 144 objects are shown in Fig. 7(A–C). The objects undergo small pose changes in this experiment in order to guarantee a very large number of valid optical flow and stereo samples for each segment so as to maximize the computational complexity. In this way it constitutes a worst-case scenario that provides an upper bound on the computational requirements. The fewer objects there are, the closer to the camera we position them so as to have a sufficient number of valid samples for the experiment.

Three internal (M-estimation) and three external (ICP) iterations were performed. The component and total times for a 6DOF tracking problem ranging from one to 160 objects are shown in Fig. 7. All times were obtained employing a single GPU of a Geforce GTX 590. As expected, the pre-processing stage (dominated by the radix sort) is the most time-consuming. A substantial increase occurs at 16 objects since a switch is made from 4 to 8 bit radix sorting to handle the segment labels exceeding 16. More efficient and gradual implementations can be used here instead [48]. The times required to compose and solve the normal equations (including CPU–GPU transfers), and for rendering, increase linearly (but slowly) with number of objects. In general this increase is controlled and we also see excellent scaling in terms of the number of samples used from 50,000 (Fig. 7D) to 500,000 (Fig. 7E). With the number of objects considered here, rendering and normal equations solving times are negligible.

A breakdown of the time required to compute the low level dense cues for a single 640×480 image frame is provided in Table I (once again using one GPU of a Geforce GTX 590). As discussed in Section III-A, four Gabor pyramids need to be computed (left, right, 2× left AR). A number of rendering steps are also required to create the stereo priors and AR images. Note that the low-level component computes dense stereo and three times optical flow altogether at ±100 Hz. Table II contains the framerates achieved by the complete tracking system (low-level and pose updates) for a number of different configurations. These times correspond to those reported in Fig. 7F. Note that the complete tracking system operates at 38 Hz when tracking 150 objects using 500,000 samples. The computational requirements of the most demanding parts of the tracking component, namely the low-level vision and radix-sorting, scale approximately linearly in the number of pixels [33], [45]. Since they are massively parallel, the tracking component can process higher resolution images faster on GPUs equipped with more processing cores.

The sparse detection component runs independently on the second GPU of the Geforce GTX 590 so that it does not affect the tracker's speed. Its estimates are employed when available and so its speed mainly determines the recovery latency in case tracking is lost. Our current implementation runs at 20 Hz. This can be increased by reducing the model size and/or using

TABLE I
PROCESSING TIMES DENSE LOW-LEVEL CUES (IN MS)

dense cues *(640×480 image size)*	
– image transfer CPU → GPU	0.4
– Gabor pyramid (4×)	3.3
– optical flow	1.5
– AR flow (2×)	3.0
– model-based stereo	1.2
– rendering	0.9
total	10.3

TABLE II
TRACKING FRAME RATES (IN HZ)

	# samples	
# objects	50,000	500,000
1	61	55
20	54	43
150	46	38

more efficient keypoints and descriptors, but at the cost of reduced accuracy. An alternative is to use FPGA-accelerated SIFT implementations [49].

IV. SYNTHETIC BENCHMARK DATASET

We have constructed an extensive synthetic benchmark dataset to evaluate pose trackers under a variety of realistic conditions. Its creation is discussed in detail next, but Fig. 11 already shows some representative examples that illustrate the large distance range, background and object variability, and the added noise and occlusions. The complete dataset is available on-line.[1] Note that we only focus on the single object case here to facilitate the comparison with alternative methods. We show many real-world examples involving multiple objects in Section V-C and *Supplemental Material Video 3*.[2]

A. Object Models

We have selected four objects from the publicly available KIT object models database [50]. Snapshots of these models are shown in Fig. 8 (*soda*, *soup*, *clown*, and *candy*) and provide a good sample of the range of shapes and textures available in the database. We also included two cube-shaped objects, one richly textured (*cube*), and the other containing only edges (*edge*).

[1]http://www.karlpauwels.com/datasets/rigid-pose/
[2]All supplemental material is available at http://www.karlpauwels.com/ieee-tcsvt-2015-supplemental-material/

Fig. 8. Object models used in the synthetic sequences.

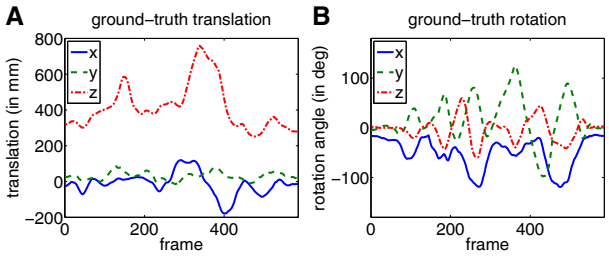

Fig. 9. Ground-truth object motion in synthetic sequences.

B. Object and Background Motion

Using the proposed system, we recorded a realistic and complex motion trace by manually manipulating an object similar to *cube*. This, possibly erroneous, motion trace was then used to generate synthetic sequences and so it is, by definition, the ground-truth object motion. The trace is shown in Fig. 9 and covers a high dynamic range, varying all six degrees of freedom. The sequence was recorded at 30 Hz and consists of 586 frames. The object's translational speed ranged from 5 mm/s to 452 mm/s, with average 137 mm/s and standard deviation 79 mm/s. The rotational speed ranged from 1 deg/s to 221 deg/s, and averaged at 81 deg/s with standard deviation 49 deg/s. The realism and complexity of the sequences is further increased by blending the rendered objects into a real-world stereo sequence recorded with a moving camera in a cluttered office environment. The camera motion only serves to prohibit the use of a background subtraction algorithm as a segmentation pre-processing step. The object pose itself is always expressed and estimated with respect to the camera. Some examples are shown in Fig. 11 but, to fully appreciate the complexity, we refer to *Supplemental Material Video 1*.

C. Noise and Occluder

To further explore the limitations of pose tracking methods, different sequences are created corrupted either by noise or an occluding object. For the noisy sequences, Gaussian noise (with σ equal to one tenth of the intensity range) is added separately to each color channel, frame, and stereo image (Fig. 11B). To obtain realistic occlusion (with meaningful motion and stereo cues), we added a randomly bouncing 3D teddy bear object to the sequence (Fig. 11C,D and *Supplemental Material Video 1*). The occlusion proportion of the *cube* object over the sequence is shown in Fig. 10. Although this differs for the left and right sequences, none of the methods evaluated here exploit this (*e.g.* our dense stereo cue is affected by *either* left or right occlusions).

D. Performance Evaluation

Pose trackers are usually evaluated by comparing the estimated to the ground-truth or approximate ground-truth pose across an entire sequence [14], [15], [18], [20]–[22]. However, once a tracker is lost, the subsequent estimates (and their errors) become irrelevant. For example, if tracking is lost early in the scene, the average error will typically be very large, but

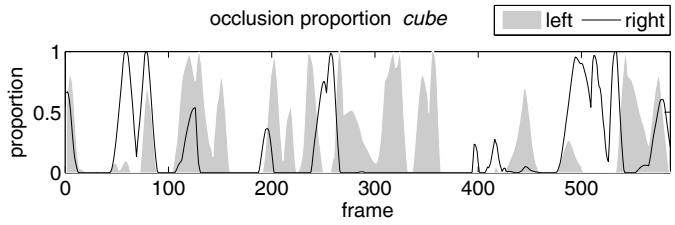

Fig. 10. Proportion occlusion in the *cube* sequence.

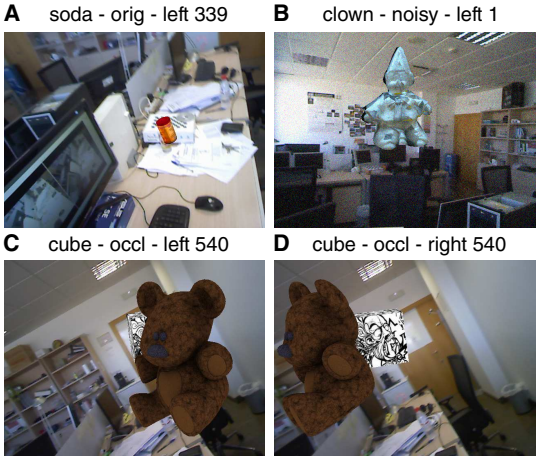

Fig. 11. Indicative samples from the different synthetic sequences illustrating (A) large distance range, (B) added noise, and (C,D) different occlusion proportions in the left and right sequences (orig = noise free; occl = occluded; number = frame index).

this doesn't mean the tracker cannot track the object in the remainder of the sequence, if re-initialized.

For this reason, we propose to instead measure the *proportion* of a sequence that can be tracked successfully. Since we use synthetic sequences, we can continuously monitor tracking accuracy and automatically reset when tracking is lost. But how to decide if tracking is lost? Rotations and translations affect an object's position in very different ways and are therefore hard to summarize into a single error. Instead we use the largest distance between corresponding vertices, $\mathbf{v}_j$, of the object transformed according to the ground-truth $(\mathrm{R}, \mathbf{t})$ and estimated pose $(\hat{\mathrm{R}}, \hat{\mathbf{t}})$:

$$e_P = \max_j \|(\hat{\mathrm{R}}\,\mathbf{v}_j + \hat{\mathbf{t}}) - (\mathrm{R}\,\mathbf{v}_j + \mathbf{t})\| \ . \qquad (28)$$

This measure is easy to interpret and is sensible from a task-based perspective. Considering for example a grasping or collision-avoidance task, it is important to understand the positional accuracy of the entire object, incorporating both its shape and the pose. When this distance exceeds a threshold (e.g. 10 mm), the tracker is reset to the ground-truth. The proportion of the sequence tracked correctly then constitutes a scalar performance measure for the entire sequence. To put this measure in perspective, a *static* error is also computed for each sequence using a 'naive tracker'. This 'tracker' simply never updates the pose. As a consequence all resets are triggered by the object motion alone and the error provides an indication of the sequence complexity. For example, in a sequence with a static object, perfect performance will be achieved.

TABLE III
TRACKING SUCCESS RATE (IN %) – STEREO AND OPTICAL FLOW
ORIG = NOISE FREE; OCCL = OCCLUDED

	soda			*edge*		
	orig	noisy	occl	orig	noisy	occl
static	53	53	53	50	50	50
stereo	77	47	42	60	51	33
optical flow	93	81	57	78	81	40
stereo+flow	100	96	64	92	93	52

V. RESULTS

A. Stereo and Optical Flow Synergy

Table III shows results on the least textured sequences *soda* and *edge*. It contains the proportion of the sequence that can be tracked correctly, as defined in the previous section, by different configurations of the proposed method either using only stereo, only optical flow, or both cues together. The results are evaluated on all three versions of the sequences, namely noise-free (*orig*), *noisy*, and occluded (*occl*). The *static* performance is around 50% everywhere, which means that, without tracking, a reset is required approximately every other frame. Due to the low texture, shape-symmetry (*soda*) and shape-planarity (*edge*), stereo-only performance is quite bad in these sequences and even below *static* in the noisy and occluded scenarios. Optical-flow-only performance is better but, when both are combined, great improvements can be observed. This highlights the importance of combining multiple cues, particularly in low-texture situations. The AR flow and sparse cues further improve the results, but these are discussed in the next section (and shown in Table V).

B. State-of-the-art Methods

The Blocks World Robotic Vision Toolbox (BLORT) [5] provides a number of object learning, recognition, and tracking components that can be combined to robustly track object pose in real-time. We only evaluate the particle-filter-based tracking module here since the recognition module is very similar to our sparse-only method. Each particle represents a pose estimate and is used to render an edge model (combining edges from geometry and texture) that is matched against edges extracted from the current image. We evaluate BLORT's tracking module with the default (real-time) setting with 200 particles and a high precision variant with 10,000 particles. Due to an inefficient rendering procedure, the current tracker implementation of BLORT can not handle models with a high vertex count. We therefore limited the geometrical complexity of the models to 800 triangles in all the sequences.

We also evaluate a state-of-the-art real-time-capable region-based tracker. The PWP3D method [20] uses a 3D geometry model to maximize the discrimination between statistical foreground and background appearance models, by directly operating on the 3D pose parameters. To ensure the best possible performance, we used very small gradient descent step sizes (0.1 degrees for rotation and 1 mm for translation). Together with a large number of iterations (100), this ensures stable convergence (although no longer in real-time). Furthermore,

TABLE IV
TRACKING ERRORS ($e_T^{\{x,y,z\}}$ AND e_P IN MM, $e_R^{\{x,y,z\}}$ IN DEGREES) AND SUCCESS RATES (SR IN %) OBTAINED ON THE SODA SEQUENCE

	soda – noise free							
	e_T^x	e_T^y	e_T^z	e_R^x	e_R^y	e_R^z	e_P	SR
sparse-and-dense	0.2	0.2	1.1	0.3	0.7	0.6	1.5	98.5
dense-only	**0.3**	**0.2**	**0.9**	**0.3**	**0.9**	**0.6**	**1.5**	**100.0**
sparse-only	0.7	0.4	3.1	1.0	1.4	0.6	4.2	61.1
part. filt. 10,000	1.2	0.7	3.9	1.3	2.8	1.3	5.7	75.5
region-based	0.5	1.1	2.2	1.9	7.2	4.8	6.2	84.4
part. filt. 200	1.3	1.0	3.9	1.9	3.1	1.8	6.1	57.5
static	3.1	1.9	3.5	1.6	2.8	1.9	6.8	53.1
	soda – noisy							
	e_T^x	e_T^y	e_T^z	e_R^x	e_R^y	e_R^z	e_P	SR
sparse-and-dense	0.9	0.7	2.3	0.8	2.0	1.9	3.8	95.9
dense-only	**0.9**	**0.6**	**2.3**	**0.8**	**2.1**	**1.9**	**3.8**	**97.1**
sparse-only	1.2	0.8	3.1	1.7	2.4	1.2	5.3	36.5
part. filt. 10,000	1.5	0.8	4.4	1.3	2.8	1.2	6.2	65.2
region-based	0.5	1.1	2.2	2.2	6.6	4.9	6.1	84.4
part. filt. 200	1.4	1.0	4.1	1.8	3.3	1.9	6.4	59.6
static	3.1	1.9	3.5	1.6	2.8	1.9	6.8	53.1
	soda – occluded							
	e_T^x	e_T^y	e_T^z	e_R^x	e_R^y	e_R^z	e_P	SR
sparse-and-dense	**0.7**	**0.5**	**1.8**	**0.9**	**1.6**	**1.1**	**3.0**	**68.3**
dense-only	0.7	0.5	1.6	0.9	1.5	1.3	2.9	67.0
sparse-only	0.8	0.4	3.2	1.1	1.5	0.8	4.3	44.0
part. filt. 10,000	1.7	1.1	3.9	1.2	3.0	1.1	6.0	53.8
region-based	0.7	1.1	2.4	1.9	7.2	5.3	6.3	44.0
part. filt. 200	1.7	1.2	4.2	1.9	3.5	2.1	6.8	45.2
static	3.1	1.9	3.5	1.6	2.8	1.9	6.8	53.1

we initialized the PWP3D method at each frame with the ground-truth color histogram of the actual (or unoccluded) frame being processed so that also inaccuracies here do not affect performance.

1) Tracking Success Rates: Table V summarizes all the results obtained with a tracking reset threshold equal to 10 mm. To better quantify the precision obtained, we have also computed the Root Mean Squared (RMS) errors, obtained on the successful frames, for the three translational ($e_T^{\{x,y,z\}}$) and three rotational ($e_R^{\{x,y,z\}}$) components of the pose (using Euler angles for the rotation components) and for the maximal distance error (28). For each condition, we highlight the best result in Table V, considering first the Success Rate (SR) and then the RMS distance error. The more detailed measures can be found in Table IV, but only for the *soda* sequence. The complete table is available at the supplemental material website.

The proposed *dense-only* tracking method obtains an SR close to 100% regardless of model shape, texture (see *edge*), or sequence noise (although e_P increases with noise as can be expected). In the occluded scenario however, it is frequently outperformed by the *sparse-only* and high quality *particle filter* methods. But, when combined with the sparse method (II-C2) the synergy of both modules is confirmed. *Sparse-and-dense* retains the excellent performance of the *dense-only* method with greatly improved robustness to occlusions, even outperforming *sparse-only* on most sequences. The slight decrease in performance of *sparse-and-dense* with respect to *dense-only* is due to the limits of the selection mechanism. At times, a less accurate sparse estimate can result in a larger AR-flow proportion. A potential improvement is to also consider the magnitude of the AR flow, but we found this to be less robust. More important though is that, unlike *dense-only*, *sparse-and-dense* also enables recovery from tracking failures, which is critical in real-world applications. The *particle filter* method performs very well provided a very large number of particles are used. In the real-time setting (200 particles) the performance is not much better than *static*. The *sparse-only* method performs badly on *soda* due to the weak texture, and fails on *edge* due to the complete lack of texture. The *region-based* tracker performs consistently well on the noise-free and noisy sequences, but fails dramatically in the presence of an occluder. Although it can handle certain types of occlusions, large failures occur when an entire side of the object contour is occluded [20].

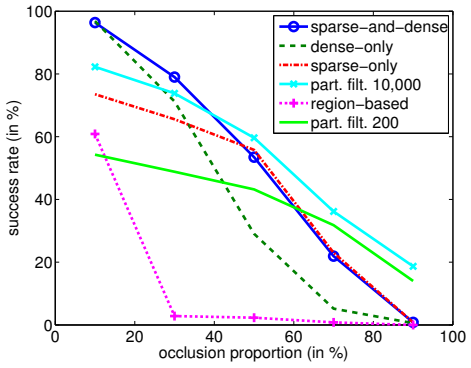

Fig. 12. Average tracking success rates obtained across all benchmarking sequences as a function of the proportion of the tracked object that is occluded.

2) Occlusion Robustness: Figure 12 shows the evolution of the tracking success rates as a function of the proportion of the object that is occluded. This graph was generated by binning all the instances where the tracking error exceeds the reset threshold according to the occlusion proportion of the object at that frame. It highlights the synergy obtained with the *sparse-and-dense* method. It retains *sparse-only*'s robustness at high occlusion proportions, while achieving a higher success rate at lower occlusion proportions than either *sparse-only* or *dense-only*. Figure 12 also illustrates the high sensitivity to occlusion of the *region-based* method, which breaks down completely between 20% and 30% occlusion. Finally, the *particle filter* methods demonstrate a slightly higher robustness at high occlusion proportions. This is because at very high occlusion levels, the robust estimation component of the proposed method will break down and the optical flow estimated at the occluder will also be used to update tracking. As discussed in Section II-C2 our reliability measure can signal such extreme occlusions, and so tracking will typically be disabled in these circumstances. See the next section and specifically Fig. 13 for examples of this behavior on real-world sequences.

C. Real-world Sequences

1) Complex Scenarios: The proposed method also yields excellent results in real-world scenarios. Some example single object results with a cluttered scene, occluders, and camera motion are shown in Fig. 13. The dense estimate is selected as winner in Fig. 13A,B. This usually occurs when the object

TABLE V
TRACKING SUCCESS RATE (IN %) – ORIG = NOISE FREE; OCCL = OCCLUDED

	soda			*soup*			*clown*			*candy*			*cube*			*edge*		
	orig	noisy	occl	orig	noisy	occl	orig	noisy	occl	orig	noisy	occl	orig	noisy	occl	orig	noisy	occl
sparse-and-dense	98.5	95.9	**68.3**	97.9	97.3	**78.3**	100.0	98.3	**77.7**	99.8	99.5	**79.5**	100.0	100.0	75.7	**96.2**	**97.4**	56.2
dense-only	**100.0**	**97.1**	67.0	**99.0**	**98.1**	69.7	**100.0**	**100.0**	68.5	**100.0**	**100.0**	72.9	**100.0**	**100.0**	69.9	**96.2**	**97.4**	56.2
sparse-only	61.1	36.5	44.0	93.0	74.0	76.5	92.0	71.1	74.1	95.4	90.4	80.0	97.6	95.9	**78.8**	0.0	0.0	0.0
part. filt. 10,000	75.5	65.2	53.8	76.9	65.8	62.5	87.5	82.4	76.4	76.7	76.4	63.9	93.2	93.7	76.2	72.4	91.3	**68.2**
region-based	84.4	84.4	44.0	96.2	95.5	44.0	96.4	89.4	43.8	83.6	83.9	39.2	83.9	74.3	37.8	84.6	83.7	38.9
part. filt. 200	57.5	59.6	45.2	46.7	53.9	40.2	56.2	62.2	48.1	45.9	48.8	40.9	53.1	53.6	38.9	63.2	62.5	49.7
static	53.1	53.1	53.1	45.0	45.0	45.0	47.1	47.1	47.1	46.4	46.4	46.4	50.2	50.2	50.2	50.2	50.2	50.2

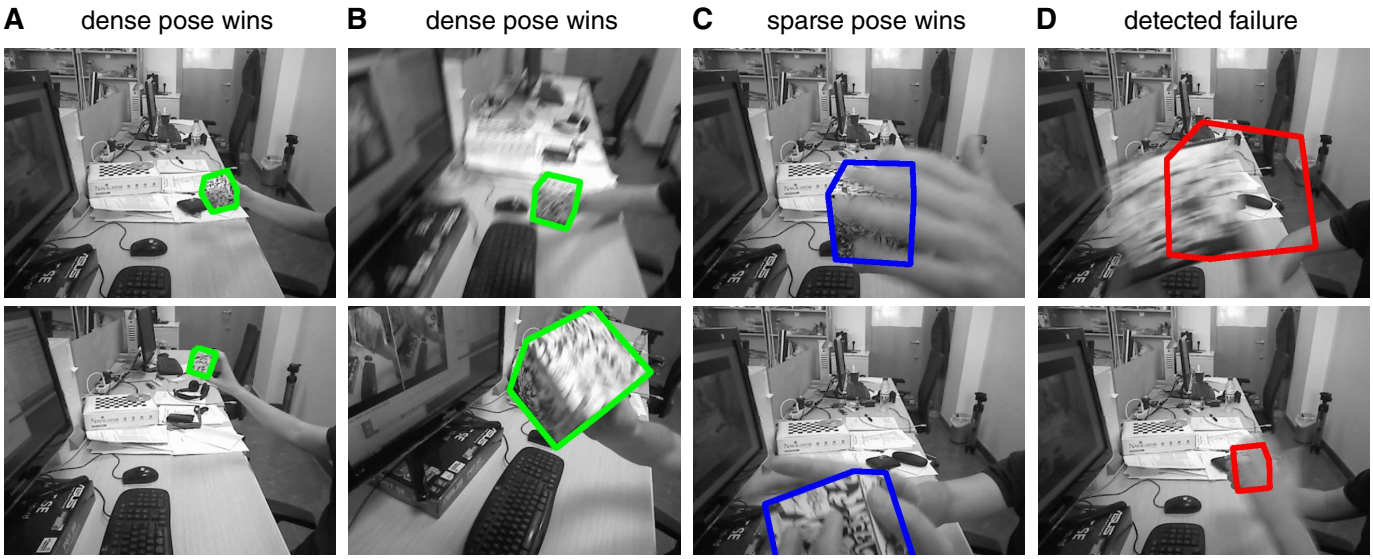

Fig. 13. Indicative real-world single object pose estimation results, showing how the dense pose is selected when (A) the object is far and/or (B) motion-blurred, how the sparse pose is selected in case of (C) strong occlusions, and how (D) failures can be detected correctly.

is far away (A) or suffering from severe motion blur (B). The sparse estimate is usually selected when only a small part of the object is visible (C). Figure 13D finally shows some tracking failures that are detected correctly by the reliability measure (proportion AR flow < 0.15). See *Supplemental Material Video 2* for more single object real-world results.

Figure 14 contains three snapshots of longer sequences contained in *Supplemental Material Video 3*. This video demonstrates the proposed method in various complex scenarios involving multiple interacting objects undergoing manipulation. The 3D models were obtained using a publicly available solution [1]. In some of these scenarios (Fig. 14A,B) we also show an image rendered from a viewpoint different from the camera, to more clearly show the precision obtained by our system. Note how in Fig. 14A,B the objects are aligned quite precisely even though we do not prevent them from intersecting each other. Our approach can handle a great variety in shape and appearance (*e.g.* the *melon* object in Fig. 14B). The method is also quite robust to inaccuracies at the modeling stage (*e.g.* due to limits of the acquisition process the bottoms of the objects are never modeled, see Fig. 14B *caserio* object, but see also Fig. 2). Occlusions between the different objects are also handled automatically through the rendering process and subsequent label assignment. Figure 14C shows how the model-based stereo handles multiple objects and occlusions.

2) Comparison to Particle Filter Method: We finally compare *sparse-and-dense* to the *particle filter* method on real-world sequences in which a single object is manipulated. Some example images are shown in Fig. 15 but the complete sequences are available as *Supplemental Material Video 4*. For the object considered here the *particle filter* method achieves near real-time performance using 200 particles (15 Hz) but requires multiple seconds per frame when using 10,000 particles (0.3 Hz). The results agree with those obtained on the synthetic benchmarking dataset. The poses estimated with 200 particles are jittery, but become much more stable with 10,000 particles. With this large number of particles they are visually similar to the estimates obtained with *sparse-and-dense*. When we increase the manipulation speed, motion blur is introduced in the scene and the *particle filter* method quickly loses track, see Fig. 15 right column. The proposed method successfully tracks the object throughout the sequences.

VI. DISCUSSION

Although highly robust, the edge-based particle filter method requires a large number of particles to achieve high accuracy. Since each particle requires a rendering step, the performance critically depends on model complexity. This method, and other related particle filter methods [26] are

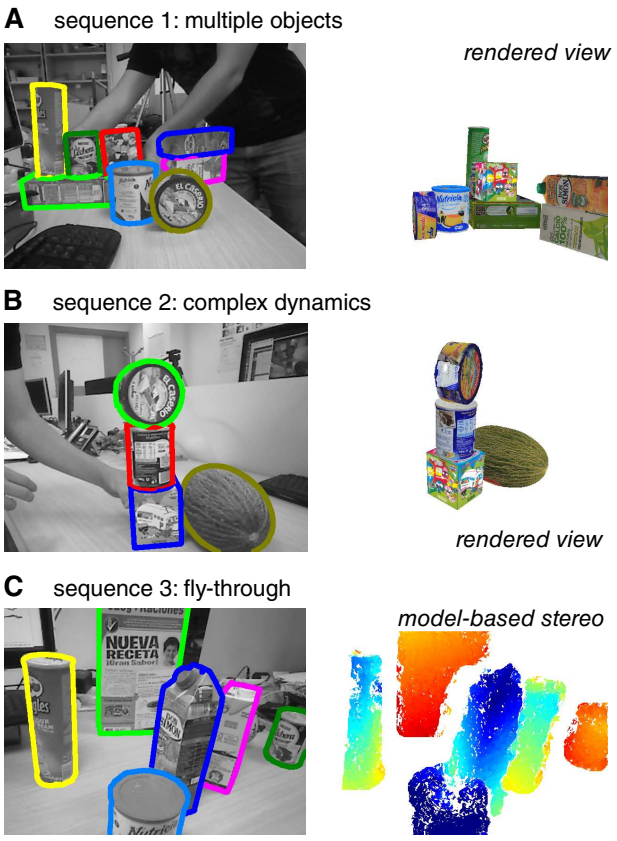

Fig. 14. Real world pose detection and tracking results involving multiple objects and complex dynamics. These images are representative snapshots of much longer sequences (see *Supplemental Material Video 3*).

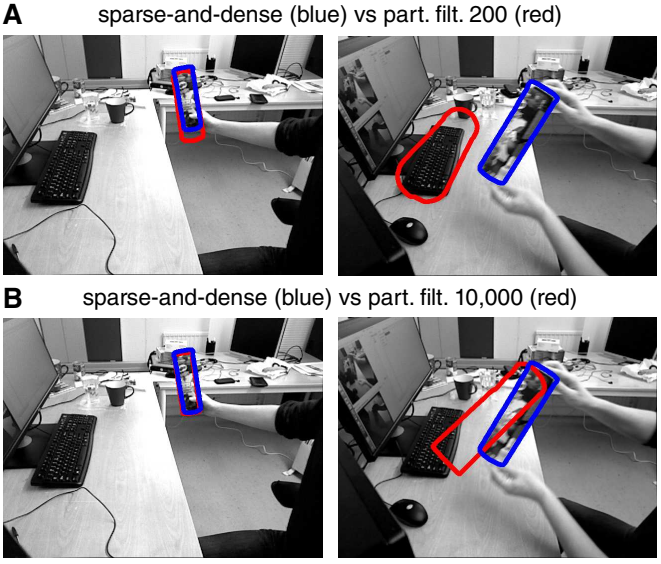

Fig. 15. Comparison between *sparse-and-dense* and the *particle filter* method on a real-world image sequence. In the left column (A to B) we see the improved accuracy that results from increasing the number of particles. In the right column we see the failure of the *particle filter* method due to high speed object motion and motion blurring. These images are representative snapshots of much longer sequences (see *Supplemental Material Video 4*).

also difficult to extend to the articulated scenario due to the increased dimensionality of the problem. The sparse keypoint-based method is highly robust to occlusions and provides excellent synergy with the dense methods proposed here. The region-based method does not require edges or texture and performs very well. It does have problems with symmetric objects, is slow to converge, and fails on certain types of occlusions. This requires the use of multiple cameras or explicit modeling of the occlusions. There is much potential for incorporating color-based (region-based) cues into the proposed method. However it is not straightforward to process these in a GPU-friendly manner.

Note that the proposed method also supports depth cues other than stereo (e.g. from a Kinect sensor), and conversely, enables for the incorporation of motion cues in current depth-only applications [2]. Current Kinect versions however do not provide the high shape detail close to the camera, nor the high framerates achieved by our model-based stereo algorithm [51].

The scalability results shown here can be considered somewhat artificial. Nonetheless, there are many situations where this approach is useful. Multiple (not necessarily overlapping) camera views can be combined to simultaneously track a very large number of objects. The same could be achieved by equipping a higher resolution sensor with a wide-angle lens. Also articulated or non-rigid objects could be decomposed or approximated by a large number of rigid components that can be jointly tracked using this approach. Specifically, it has been shown how the parts can be considered as separate rigid objects and the constraints enforced later [13], [52]. Additional constraints can be obtained from physics simulations and incorporated in a temporal filtering framework considering also (multi-)object persistence, manipulator feedback, etc.

VII. Conclusion

We have presented a novel model-based multi-cue approach for simultaneously tracking the 6DOF poses of a very large number of rigid objects that exploits dense motion and stereo cues, sparse keypoint features, and feedback from the modeled scene to the cue extraction. The method is inherently parallel and efficiently implemented using GPU acceleration. We have introduced an evaluation methodology and benchmark dataset specifically for this problem. Using this dataset we have shown improved accuracy, robustness, and speed of the proposed method as compared to state-of-the-art real-time-capable methods.

Acknowledgment

The authors gratefully acknowledge the European FP7 project RoboHow (FP7-ICT-288533) and the Spanish National Project NEUROPACT (TIN2013-47069-P). The GPU used for this research was donated by the NVIDIA Corporation.

References

[1] Autodesk, "123D Catch," http://www.123dapp.com/catch/, 2014.
[2] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: real-time dense surface mapping and tracking," in *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, Oct. 2011, pp. 127–136.

[3] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3D object modeling," *Int. J. Robot. Res.*, 2011.

[4] L. Vacchetti, V. Lepetit, and P. Fua, "Stable real-time 3D tracking using online and offline information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 10, pp. 1385 –1391, Oct. 2004.

[5] T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, and M. Vincze, "BLORT - The Blocks World Robotic Vision Toolbox," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2010.

[6] A. Collet, M. Martinez, and S. S. Srinivasa, "The MOPED framework: Object recognition and pose estimation for manipulation," *Int. J. Robot. Res.*, vol. 30, no. 10, pp. 1284–1306, Apr. 2011.

[7] Q. Hao, R. Cai, Z. Li, L. Zhang, Y. Pang, F. Wu, and Y. Rui, "Efficient 2D-to-3D correspondence filtering for scalable 3D object recognition," in *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, June 2013.

[8] S. Holzer, S. Hinterstoisser, S. Ilic, and N. Navab, "Distance transform templates for object detection and pose estimation," in *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2009, pp. 1177–1184.

[9] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *Proc. Int. Conf. on Computer Vision*, 2011, pp. 858–865.

[10] P. Besl and N. McKay, "A method for registration of 3D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, pp. 239–256, 1992.

[11] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes," in *Proc. Asian Conf. on Computer Vision*. Springer, 2013, pp. 548–562.

[12] J. Liebelt and C. Schmid, "Multi-view object class detection with a 3D geometric model," in *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2010, pp. 1688–1695.

[13] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 932–946, 2002.

[14] V. Kyrki and D. Kragic, "Tracking rigid objects using integration of model-based and model-free cues," *Mach. Vision Appl.*, vol. 22, pp. 323–335, 2011.

[15] M. Pressigout and E. Marchand, "Real-time hybrid tracking using edge and texture information," *Int. J. Robot. Res.*, vol. 26, pp. 689–713, 2007.

[16] M. Pressigout, E. Marchand, and E. Memin, "Hybrid tracking approach using optical flow and pose estimation," in *Proc. IEEE Int. Conf. on Image Processing*, Oct. 2008, pp. 2720–2723.

[17] P. Azad, D. Munch, T. Asfour, and R. Dillmann, "6-DoF model-based tracking of arbitrarily shaped 3D objects," in *Proc. IEEE Int. Conf. on Robotics and Automation*, May 2011, pp. 5204–5209.

[18] C. Choi and H. Christensen, "Robust 3D visual tracking using particle filtering on the special euclidean group: A combined approach of keypoint and edge features," *Int. J. Robot. Res.*, vol. 31, no. 4, pp. 498–519, 2012.

[19] C. Teuliere, E. Marchand, and L. Eck, "Using multiple hypothesis in model-based tracking," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2010, pp. 4559–4565.

[20] V. Prisacariu and I. Reid, "PWP3D: Real-time segmentation and tracking of 3D objects," *Int. J. Comput. Vision*, vol. 98, pp. 335–354, 2012.

[21] T. Brox, B. Rosenhahn, J. Gall, and D. Cremers, "Combined region and motion-based 3D tracking of rigid and articulated objects," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 3, pp. 402–415, 2010.

[22] A. Petit, E. Marchand, and K. Kanani, "A robust model-based tracker combining geometrical and color edge information," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013.

[23] Q. Wang, W. Zhang, X. Tang, and H.-Y. Shum, "Real-time bayesian 3-D pose tracking," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 12, pp. 1533–1541, Dec 2006.

[24] A. Milan, S. Roth, and K. Schindler, "Continuous energy minimization for multitarget tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 58–72, 2014.

[25] P. Azad, T. Asfour, and R. Dillmann, "Accurate shape-based 6-DoF pose estimation of single-colored objects," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Oct. 2009, pp. 2690–2695.

[26] C. Choi and H. Christensen, "RGB-D object tracking: A particle filter approach on GPU," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013, pp. 1084–1091.

[27] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Jun. 2011, pp. 1297–1304.

[28] C. Teuliere and E. Marchand, "Direct 3D servoing using dense depth maps," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, pp. 1741–1746.

[29] A. Talukder and L. Matthies, "Real-time detection of moving objects from moving vehicles using dense stereo and optical flow," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 4, Sep. 2004, pp. 3718–3725 vol.4.

[30] T. Komuro, I. Ishii, M. Ishikawa, and A. Yoshida, "A digital vision chip specialized for high-speed target tracking," *IEEE Trans. Electron Devices*, vol. 50, no. 1, pp. 191–199, 2003.

[31] K. Pauwels, L. Rubio, J. Diaz Alonso, and E. Ros, "Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues," in *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Portland, June 2013.

[32] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[33] K. Pauwels, M. Tomasi, J. Díaz, E. Ros, and M. Van Hulle, "A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 999–1012, 2012.

[34] M. Tomasi, M. Vanegas, F. Barranco, J. Daz, and E. Ros, "Massive parallel-hardware architecture for multiscale stereo, optical flow and image-structure computation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 2, pp. 282–294, 2012.

[35] C. Wu, "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)," http://cs.unc.edu/~ccwu/siftgpu, 2007.

[36] S. Sabatini, G. Gastaldi, F. Solari, K. Pauwels, M. Van Hulle, J. Díaz, E. Ros, N. Pugeault, and N. Krüger, "A compact harmonic code for early vision based on anisotropic frequency channels," *Comput. Vis. Image Und.*, vol. 114, no. 6, pp. 681–699, 2010.

[37] K. Pauwels and M. Van Hulle, "Optic flow from unstable sequences through local velocity constancy maximization," *Image Vision Comput.*, vol. 27, no. 5, pp. 579–587, 2009.

[38] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[39] G. Blais and M. Levine, "Registering multiview range data to create 3D computer objects," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 8, pp. 820–824, 1995.

[40] C. Yang and G. Medioni, "Object modelling by registration of multiple range images," *Image Vision Comput.*, vol. 10, no. 3, pp. 145–155, 1992.

[41] H. C. Longuet-Higgins and K. Prazdny, "The interpretation of a moving retinal image," *P. Roy. Soc. B-Biol. Sci.*, vol. 208, pp. 385–397, 1980.

[42] F. Mosteller and J. Tukey, *Data analysis and regression: A second course in statistics*. Mass.: Addison-Wesley Reading, 1977.

[43] V. Lepetit and P. Fua, "Monocular model-based 3D tracking of rigid objects," *Foundations and Trends in Computer Graphics and Vision*, vol. 1, pp. 1–89, 2005.

[44] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[45] J. Hoberock and N. Bell, "Thrust: A parallel template library," 2010, version 1.7.0. [Online]. Available: http://thrust.github.io/

[46] K. Pauwels, N. Krüger, M. Lappe, F. Wörgötter, and M. Van Hulle, "A cortical architecture on parallel hardware for motion processing in real time," *J. Vision*, vol. 10, no. 10, 2010.

[47] S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri, "An efficient algorithm for the approximate median selection problem," in *Algorithms and Complexity*, ser. Lecture Notes in Computer Science, G. Bongiovanni, R. Petreschi, and G. Gambosi, Eds. Springer Berlin Heidelberg, Jan. 2000, no. 1767, pp. 226–238.

[48] D. Merrill and A. Grimshaw, "High performance and scalable radix sorting: A case study of implementing dynamic parallelism for GPU computing," *Parallel Processing Letters*, vol. 21, no. 02, pp. 245–272, 2011.

[49] J. Jiang, X. Li, and G. Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 7, pp. 1209–1220, 2014.

[50] A. Kasper, Z. Xue, and R. Dillmann, "The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics," *Int. J. Robot. Res.*, vol. 31, no. 8, pp. 927–934, 2012.

[51] Wikipedia, "Kinect," 2014.

[52] K. Pauwels, L. Rubio, and E. Ros, "Real-time model-based articulated object pose detection and tracking with variable rigidity constraints," in *Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Columbus, Ohio, 2014.

Karl Pauwels received the M.Sc. degree in Commercial Engineering, the M.Sc. degree in Artificial Intelligence, and the Ph.D. degree in Medical Sciences from the Katholieke Universiteit Leuven, Belgium. He is currently a postdoc at the Computer Vision and Active Perception lab of the Royal Institute of Technology Stockholm, Sweden (KTH). His main research interest is real-time computer and robot vision in the context of autonomous navigation and dexterous manipulation of complex objects.

Leonardo Rubio received his M.Sc. in Computer and Network Engineering in 2009 and the Ph.D. degree in Computer Science in 2014, both from the University of Granada, Spain. He has been involved in the European Project TOMSY. He is currently working at Fuel3D Technologies LTD as a computer vision researcher. His main research interests are object detection, tracking and 3D reconstruction.

Eduardo Ros received the Ph.D. degree in 1997 from the University of Granada. He is currently Full Professor at the Department of Computer Architecture and Technology at the same University. He is currently the responsible researcher at the University of Granada of two European projects related with bio-inspired processing schemes and real-time image processing. His research interests include hardware implementation of digital circuits for real time processing in embedded systems and high performance computer vision. His lab has been mainly focused on real-time processing schemes and implementations for different application fields such as automobile, robotics, bio-medicine, surveillance, etc.