SimTrack: A Simulation-based Framework for Scalable Real-time Object Pose Detection and Tracking

Karl Pauwels and Danica Kragic

Abstract— **We propose a novel approach for real-time object pose detection and tracking that is highly scalable in terms of the number of objects tracked and the number of cameras observing the scene. Key to this scalability is a high degree of parallelism in the algorithms employed. The method maintains a single 3D simulated model of the scene consisting of multiple objects together with a robot operating on them. This allows for rapid synthesis of appearance, depth, and occlusion information from each camera viewpoint. This information is used both for updating the pose estimates and for extracting the low-level visual cues. The visual cues obtained from each camera are efficiently fused back into the single consistent scene representation using a constrained optimization method. The centralized scene representation, together with the reliability measures it enables, simplify the interaction between pose tracking and pose detection across multiple cameras. We demonstrate the robustness of our approach in a realistic manipulation scenario.**

We publicly release this work as a part of a general ROS software framework for real-time pose estimation, *SimTrack*, that can be integrated easily for different robotic applications.

I. Introduction

Dexterous manipulation requires fast and precise pose information of the manipulated objects. Real-time visual perception, specifically model-based pose tracking, can greatly contribute to obtaining such information, and is in fact critical in uncontrolled environments where interaction with humans is required.

To ensure accurate pose tracking throughout the interaction, the placement of the camera is very important. The commonly used combined color/depth (RGB-D) cameras are frequently head-mounted. This provides a good overview of the scene and respects the minimal distance from the scene required to obtain a valid depth signal. However, it makes it more difficult to track the pose of small objects. The end effector also frequently occludes parts of the scene or the interaction in a head-mounted configuration. Alternatively, in visual servoing scenarios, the camera is often mounted close to or on the end effector (eye-in-hand). Although control is simplified, only local effects are considered and the overview of the scene is lost.

Most of these problems can be overcome using multiple cameras. By combining arm and head cameras, the field of view can be modified dynamically thus greatly increasing the likelihood of at least one camera perceiving the object. In addition, if an object is visible in multiple cameras, visual features can be combined across cameras to increase localization accuracy, *cf.* wide baseline stereo. In most current approaches however, incorporating multiple cameras greatly increases computational requirements, which excludes their use in real-time applications. To counter this, we propose a scalable method that can detect and track in real-time the pose of multiple rigid objects from multiple cameras attached to moving robotic manipulators. The method is robust to calibration errors and exploits both appearance and depth information, depending on availability.

We present three main contributions:

1) our approach exploits a high degree of parallelism to integrate dense visual cues with a highly detailed simulated model of the scene in a *scalable* manner,
2) we achieve an increased accuracy in pose estimation using *multiple cameras* for a complex scene containing *multiple objects*,
3) we release the source code as a unified ROS software framework for pose estimation: *SimTrack*[1].

In the next section, we discuss our contributions with respect to the state-of-the-art in this area. We then explain our multi-camera pose tracking approach and how it interacts with pose detection. This is followed by an experimental evaluation of the scalability and accuracy achieved by the method, together with a demonstration in a real-world stacking scenario. We conclude by pointing out several research avenues that can benefit from our approach.

A. Related Work

In the visual servoing literature it is common to rigidly attach cameras to robotic manipulators [1]. Recent extensions also consider multiple cameras simultaneously [2]. Unlike the template-based target alignment considered in the visual servoing literature, we are concerned here with estimating the full six degrees-of-freedom (DOF) pose of multiple objects, and keeping it consistent across multiple moving cameras. Multi-camera tracking has been explored previously [3], [4] but relying only on edge-like visual cues and not considering the interaction with object pose *detection*, the presence of a robot in the scene, or having the cameras move with respect to each other. Simulated models of the scene are also used in more recent sampling-based tracking approaches, but these methods require heuristic optimization techniques to scale to larger numbers of degrees of freedom [5].

The authors are with the Computer Vision and Active Perception Lab, Center for Autonomous Systems, School of Computer Science and Communication, KTH Royal Institute of Technology, Stockholm, Sweden, `{kpauwels,dani}@kth.se`. The authors gratefully acknowledge the European FP7 project RoboHow (FP7-ICT-288533). The GPUs used for this research were donated by the NVIDIA Corporation.

[1]`https://github.com/karlpauwels/simtrack`

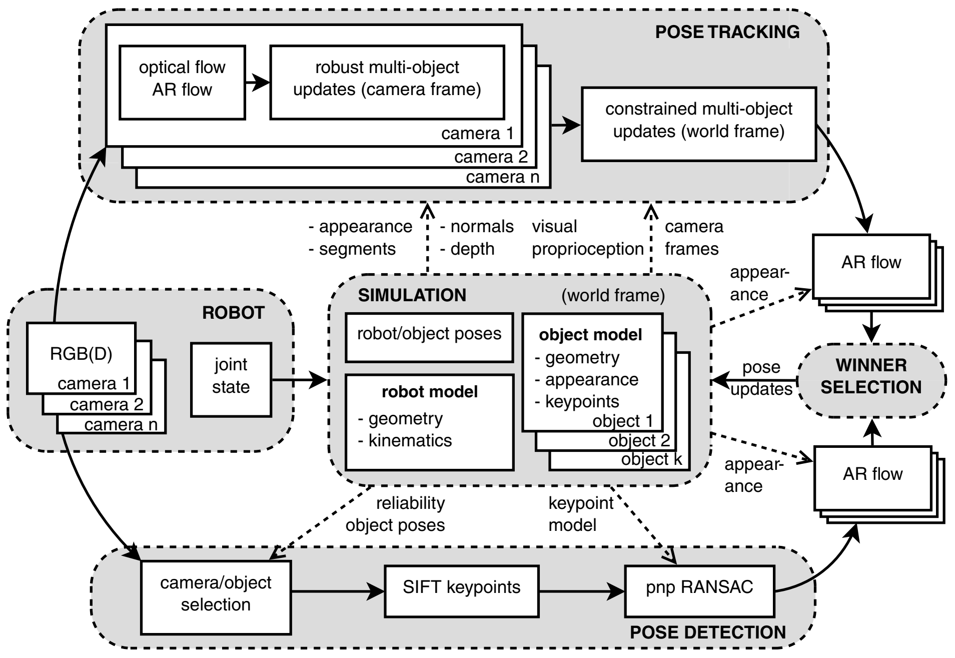

Fig. 1. Method overview (AR = augmented reality, pnp = perspective-n-point).

The work presented here naturally extends our previous work on real-time pose estimation [6] to the multi-camera case. A variety of multi-object pose detection methods are available [7]–[9], some also explicitly considering the multi-camera scenario [7], [10]. The main advantages of our work are the high tracking speed and the simpler integration of detected and tracked poses.

Calibration is an important issue in a system where multiple cameras are rigidly attached to moving robotic manipulators [11]. Calibration errors can never be completely eliminated, and should instead be corrected on-line [12]. Although this aspect is outside the scope of this work, we do demonstrate a high robustness to such calibration errors. This makes our method a clear candidate for on-line calibration.

A number of open source software frameworks exist for pose estimation and tracking, most notably ARToolkit [13], BLORT [14], and ViSP [15]. None of these however maintain a scene representation with robot, objects, and multiple cameras. They also do not exhibit the performance scaling provided by the *SimTrack* framework.

II. Methodology

Our approach relies on a continuous real-time interaction between visual simulation and visual perception, guided by a detailed 3D representation of the tracked objects and robot in world coordinates, see Fig. 1. This simulation interacts with perception, pose tracking, and pose detection by synthesizing multiple moving viewpoints together with the corresponding visual proprioceptive signals.

A. Scene Simulation

The appearance and geometry of the scene are represented as textured meshes. The object meshes are obtained using 3D object reconstruction software [16] and the robot mesh is specified using the unified robot description format (URDF). The object meshes are also augmented with SIFT features [17] extracted from different viewpoint renderings of the models. We use a graphics rendering engine to maintain a single representation of the scene in world coordinates [18]. This representation is continuously updated based on the estimated poses and the robot's joint state. Custom OpenGL shader-code allows us to rapidly synthesize the appearance, segments, depth, and normals from arbitrary viewpoints, while correctly accounting for occlusions between objects and robot. An example of the information synthesized for one viewpoint is shown in Fig. 2E–H.

B. Sensor Data and Visual Features

We use ROS to obtain sensory data from a Willow Garage PR2 robot [19]. Specifically we stream 640×480 images from the left and right arm cameras, one 640×480 RGB-D image from the head-mounted Kinect, and the joint angles. The depth from the Kinect sensor is mapped to the RGB image. Figure 2A–D shows an example of the data received from the cameras. The signals are synchronized at 30 Hz, compressed, and send to a remote system equipped with two NVIDIA Geforce GTX Titan GPUs.

We extract the same low level visual cues as in our previous work, namely dense optical flow, dense augmented reality (AR) flow, and SIFT features [6]. AR flow represents the motion between the currently hypothesized appearance of the scene as generated by the rendering engine, see Fig. 2E, and the observed image, see Fig. 2A. This cue is critical as it serves the dual purpose of countering drift in tracking and enabling reliability evaluation [20], see Section II-D. The dense optical and AR flow are estimated using multiscale

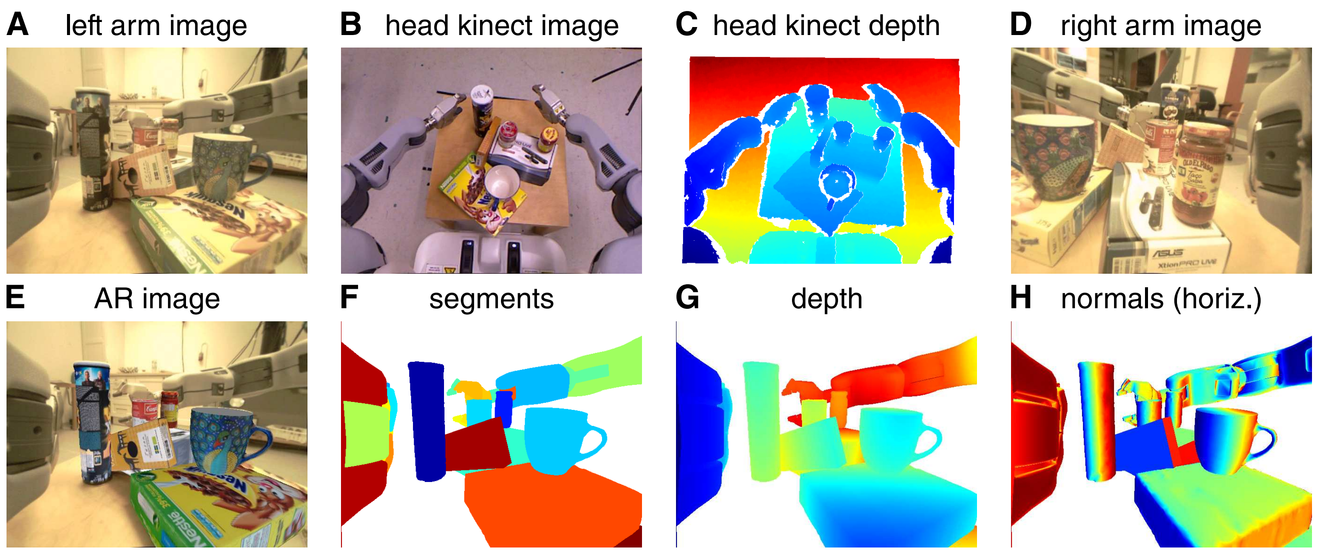

Fig. 2. Camera inputs (A–D) and scene representation (E–H) corresponding to the left arm camera.

Gabor phase-based algorithms [21].

C. Object Pose Tracking

We previously used the approach by Drummond & Cipolla [4] to extend our rigid pose estimation work to the articulated case [22]. Here we present a similar extension to the case of multiple moving cameras. Essential to the scalability of our method is maintaining the pose of each object separately in each camera frame, *i.e.* with redundant degrees of freedom. As we will show in this section, it enables for separating the expensive computation of the pose updates in each camera frame from the enforcement of the multi-camera constraints on these pose updates.

1) Single Camera: The goal of pose tracking is recovering the rigid rotation and translation that transforms each model point $\mathbf{m} = [m_x,\, m_y,\, m_z]^\top$ at time t into point $\mathbf{m}'$ at time $t+1$ in accordance with the observed visual cues:

$$\mathbf{m}' = \mathbf{R}\,\mathbf{m} + \mathbf{t}\ , \tag{1}$$

with $\mathbf{R}$ the rotation matrix and $\mathbf{t} = [t_x,\, t_y,\, t_z]^\top$ the translation vector. By approximating the rotation matrix this can be simplified to:

$$\mathbf{m}' \approx (\mathbf{I} + [\boldsymbol{\omega}]_\times)\,\mathbf{m} + \mathbf{t}\ , \tag{2}$$

with $\boldsymbol{\omega} = [\omega_x,\, \omega_y,\, \omega_z]^\top$ the rotation axis and angle, and $[\boldsymbol{\omega}]_\times$ the skew-symmetric matrix corresponding to the vector cross-product. The point-to-plane iterative closest point error [23] provides a linearized relation between the observed 3D point $\mathbf{d}' = [d'_x,\, d'_y,\, d'_z]^\top$ that corresponds to $\mathbf{m}'$, and the required pose change:

$$e_d(\mathbf{t},\boldsymbol{\omega}) = \sum_i \left(\left[(\mathbf{I} + [\boldsymbol{\omega}]_\times)\,\mathbf{m}_i + \mathbf{t} - \mathbf{d}'_i \right] \cdot \mathbf{n}_i \right)^2\ , \tag{3}$$

where $\mathbf{n}_i$ is the normal vector synthesized in the camera frame, see Fig. 2H. The correspondences are obtained using projective data association. If we consider an infinitesimal rather than discrete pose change, (2) corresponds to the differential motion equation of classical mechanics:

$$\dot{\mathbf{m}} = [\boldsymbol{\omega}]_\times \mathbf{m} + \mathbf{t}\ , \tag{4}$$

with $\dot{\mathbf{m}}$ the 3D motion of $\mathbf{m}$. After applying perspective projection with focal length f, and assuming for simplicity that the nodal point projects to the image center, the optical flow $\dot{\mathbf{x}} = [\dot{x}, \dot{y}]^\top$ can be described as [24]:

$$\dot{x} = \frac{(f\,t_x - x\,t_z)}{m_z} - \frac{x\,y}{f}\,\omega_x + (f + \frac{x^2}{f})\,\omega_y - y\,\omega_z\ , \tag{5}$$

$$\dot{y} = \frac{(f\,t_y - y\,t_z)}{m_z} - (f + \frac{y^2}{f})\,\omega_x + \frac{x\,y}{f}\,\omega_y + x\,\omega_z\ . \tag{6}$$

Note that we obtain the depth m_z by rendering the model at the current pose estimate. This allows the approach to also work in monocular configurations. Since we have two sources of pixel motion, the optical flow $\mathbf{o} = [o_x,\, o_y]^\top$ and AR flow $\mathbf{a} = [a_x,\, a_y]^\top$, we have two error functions:

$$e_o(\mathbf{t},\boldsymbol{\omega}) = \sum_i \|\dot{\mathbf{x}}_i - \mathbf{o}_i\|^2\ , \tag{7}$$

$$e_a(\mathbf{t},\boldsymbol{\omega}) = \sum_i \|\dot{\mathbf{x}}_i - \mathbf{a}_i\|^2\ . \tag{8}$$

Both the linearized point-to-plane distance in the depth case, e_d, and the differential motion constraint in the optical and AR flow case, $e_{\{o,a\}}$, now provide linear constraints on the same rigid motion representation $(\mathbf{t}, \boldsymbol{\omega})$. Therefore their combined error can also be expressed with a linear error function:

$$E(\boldsymbol{\alpha}) = (\mathbf{F}\boldsymbol{\alpha} - \mathbf{d})^\top (\mathbf{F}\boldsymbol{\alpha} - \mathbf{d})\ , \tag{9}$$

where the rigid motion parameters are stacked for convenience, $\boldsymbol{\alpha} = \begin{pmatrix} \mathbf{t} \\ \boldsymbol{\omega} \end{pmatrix}$, and $\mathbf{F}$ and $\mathbf{d}$ are obtained by gathering the sensor data according to (3), (7), and (8). This can be solved in the least-squares sense using the normal equations:

$$\mathbf{F}^\top \mathbf{F} \boldsymbol{\alpha} = \mathbf{F}^\top \mathbf{d} \ . \tag{10}$$

Since the normal equations can be composed independently for each object and camera, a high degree of parallelism is introduced in the approach.

2) Multiple Static Cameras: In the presence of noise and calibration errors, the solutions to the normal equations are not guaranteed to be consistent across multiple cameras. Drummond and Cipolla [4] pointed out that the normal equations can be used to evaluate the increased error for a suboptimal pose update. Specifically, if an unconstrained velocity update $\boldsymbol{\alpha}$ is modified into $\boldsymbol{\beta}$, the sum-squared error changes according to $(\boldsymbol{\beta} - \boldsymbol{\alpha})^\top \mathbf{C} \, (\boldsymbol{\beta} - \boldsymbol{\alpha})$, with $\mathbf{C} = \mathbf{F}^\top \mathbf{F}$. They also showed how equality constraints can be enforced between pose updates obtained in different coordinate frames. The updates can be transformed to a shared coordinate frame, in our case the world frame, using the adjoint transform that corresponds to the current camera frame. This frame is derived from the robot's joint state. The adjoint transformation is given by [25]:

$$\mathcal{T} = \mathrm{Ad}(\mathbf{T}) = \mathrm{Ad}\left(\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{R} & [\mathbf{t}]_\times \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \ . \tag{11}$$

Our problem can then be formulated as a constrained optimization problem that aims to minimize the error increase in the original camera-centric optimizations, while satisfying the inter-camera constraints. Concretely, we aim to minimize the following cost:

$$\sum_{c \in \{h,l,r\}} (\boldsymbol{\beta}_c - \boldsymbol{\alpha}_c)^\top \mathbf{C}_c \, (\boldsymbol{\beta}_c - \boldsymbol{\alpha}_c) \ , \tag{12}$$

where h, l, r refers to the head-mounted Kinect, left arm, and right arm camera respectively. An easy way to express the camera relationships is to construct a hierarchy with one camera arbitrarily chosen as root. If we select the head-mounted Kinect as root here (note that this choice does not affect the results), two sets of constraints need to be enforced: one between the Kinect and the left arm camera (16), and one between the Kinect and the right arm camera (17). The constrained optimization problem can be solved by introducing Lagrange multipliers for each constraint, namely $\boldsymbol{\lambda}_{h,l}$ and $\boldsymbol{\lambda}_{h,r}$. Setting the partial derivatives of the Lagrange system to zero then yields the following linear system of equations:

$$\begin{aligned}
2\mathbf{C}_h \boldsymbol{\beta}_h + \mathcal{T}_h^\top \boldsymbol{\lambda}_{h,l} + \mathcal{T}_h^\top \boldsymbol{\lambda}_{h,r} &= 2\mathbf{C}_h \boldsymbol{\alpha}_h && (13) \\
2\mathbf{C}_l \boldsymbol{\beta}_l - \mathcal{T}_l^\top \boldsymbol{\lambda}_{h,l} &= 2\mathbf{C}_l \boldsymbol{\alpha}_l && (14) \\
2\mathbf{C}_r \boldsymbol{\beta}_r - \mathcal{T}_r^\top \boldsymbol{\lambda}_{h,r} &= 2\mathbf{C}_r \boldsymbol{\alpha}_r && (15) \\
\mathcal{T}_h \boldsymbol{\beta}_h - \mathcal{T}_l \boldsymbol{\beta}_l &= \mathbf{0}_6 && (16) \\
\mathcal{T}_h \boldsymbol{\beta}_h - \mathcal{T}_r \boldsymbol{\beta}_r &= \mathbf{0}_6 \ . && (17)
\end{aligned}$$

Solving this system yields the constrained pose updates $\boldsymbol{\beta}_{\{h,l,r\}}$. This approach fuses the available information according to the number of motion and depth measurements available in each view. This can be weighted differently if required depending on task constraints. Objects that are not visible in all cameras do not require any modification. To increase robustness, we apply an iteratively reweighed least squares procedure based on Tukey's bisquare function [26] to the normal equations (10) before fusing them.

3) Moving Cameras and Non-linearity: So far we have assumed static cameras. Camera motion needs to be dealt with carefully since it results in image motion, referred to as visual proprioception, that should not be incorporated in the object pose updates. An easy way to account for this is to synthesize this camera-motion-induced proprioceptive optical flow and subtract it from the observed optical and AR flow *before* constructing the normal equations. The normal equations can then be fused across the cameras as if each camera was static.

A similar approach can be used to account for the nonlinearity of the problem. Since both the point-to-plane and differential motion constraints are linearized versions of the actual constraints, and since the correspondences are obtained using projective data association, the optimization needs to be iterated a number of times (set to three in the remainder). At each iteration the scene is re-rendered according to the updated pose. The component of the object motion that was already explained by the previous iteration pose update is then subtracted from the optical and AR flow together with the component due to visual proprioception.

Representing the object pose in world coordinates at time t as $\mathbf{T}^o_t$, the object poses at times 1 and 2 are related by $\Delta\mathbf{T}^o$:

$$\mathbf{T}^o_2 = \Delta\mathbf{T}^o \, \mathbf{T}^o_1 \ . \tag{18}$$

Using our current best estimate of $\Delta\mathbf{T}^o$ we can synthesize the optical flow that results from the camera and object motion as follows. Since we have a detailed simulation of the scene, our initial object pose estimate $\mathbf{T}^o_1$ can be used to generate the model depth at each pixel for a specific camera viewpoint, see Fig. 2G. Note that this correctly accounts for occlusions. From this depth we construct the visible 3D model points in the camera coordinate frame at time 1, $\mathbf{m}^c_1$. The object and camera motion now combine to arrive at the transformed model points in camera coordinates at time 2:

$$\mathbf{m}^c_2 = \mathbf{T}_2^{-1} (\Delta\mathbf{T}^o) \mathbf{T}_1 \, \mathbf{m}^c_1 \ , \tag{19}$$

where $\mathbf{T}_t$ depicts the camera's world frame pose at time t. Reading this backwards, we first transform the model points to the world frame at time 1, apply the object motion, and transform the result back to the camera frame at time 2. The synthetic optical flow is now simply obtained as the difference of $\mathbf{m}^c_2$ and $\mathbf{m}^c_1$ projected in the camera image.

D. Combined Object Pose Detection and Tracking

To detect the 6DOF object pose from a single input image, we match SIFT keypoints extracted from that image, to a 3D model codebook constructed in an initial training stage. A monocular perspective-n-point (pnp) pose estimator, robustified through RANSAC, provides a pose estimate [27].

Since the detected poses tend to be less accurate than the tracked poses, we do not merge them continuously, but instead *select* the most reliable pose [20]. This allows the system to recover from tracking failures and initialize the pose of objects entering the scene. As in our previous work, we use the proportion of valid AR flow in the visible object region as reliability measure. This indicates how well the appearance of the model rendered at the pose estimate matches the actual image. The validity of the estimated flow is determined using a forward/backward consistency check. Note that the object region over which this proportion is evaluated correctly ignores parts of the object that are occluded by other tracked objects, see Fig. 2F.

Since we favor accuracy over speed in the detection stage, we only focus on one object at a time when matching. To simplify the pnp-RANSAC-based pose estimation, we also consider one camera at a time. As a result, detection latency increases linearly with the number of objects. Note that this does not affect tracking itself, but only the recovery from tracking failures and the initialization speed. For simplicity, we randomly select the detector's object and camera here, although this selection can just as easily be biased towards the least reliable, or currently undetected objects.

By focusing on one object at a time, deciding between the tracker or detector hypothesis becomes straightforward. We first ensure that the detector does not negatively affect the reliability of any reliably tracked object in any camera. We require the following set of conditions to hold:

$$\forall(c,o) \in \{(\tilde{c},\tilde{o}) : r^t_{\tilde{c},\tilde{o}} > \tau_r\} : r^d_{c,o} \geq (r^t_{c,o} - \epsilon) \ , \qquad (20)$$

with $r^{\{t,d\}}_{c,o}$ the tracker (t) and detector (d) reliability for object o in camera c, τ_r the reliability threshold (set to 0.15 in the remainder), and ϵ a small allowed decrease in reliability (0.1) to handle noise. The detected object pose is then accepted if the highest detection reliability for that object, acquired across all cameras, exceeds an object introduction threshold τ_i (set to 0.30 in the remainder):

$$\max_c(r^d_{c,o}) > \tau_i \ . \qquad (21)$$

Note that this procedure considers all interaction effects with the other objects in all cameras. Its simplicity results from introducing only one object at a time, and from the 3D simulation that can generate reliability measures in each viewpoint that correctly account for occlusions.

For efficiency reasons, AR flow is computed once based on the tracker and once based on the detector hypothesis for each camera, as opposed to the three times that was shown in Fig. 1 for the sake of clarity. The AR flow-based winner selection is performed on the newly arriving frames, rather than at the final stage.

III. Experiments

This section contains a series of experiments to evaluate our methodology. We first examine how computation times scale with image size, number of cameras used, and number of objects in the scene. We then evaluate quantitatively how the number of cameras used affects tracking accuracy. Finally, we show real-world pose estimation results on a cluttered scene and in an object stacking scenario.

TABLE I
Low-level vision computation time (in ms)

	image size		
stage	640 × 480	3 × 640 × 480	1920 × 1080
image copy	0.2	0.6	1.4
Gabor (3×)	1.3	3.2	6.5
flow (3×)	2.8	7.4	15.4
total	4.3	11.2	23.3

TABLE II
Unconstrained pose updates computation time (in ms)

	image size / # segments			
	3 × 640 × 480		1920 × 1080	
# measurements	3	120	3	120
50,000	7.6	8.7	13.2	14.6
500,000	11.6	12.2	17.4	18.4
1,000,000	15.8	16.2	21.6	22.7

A. Scalability

The proposed method scales both in terms of number of cameras and number of objects. Imagery from multiple cameras can be handled in parallel since the object poses are first estimated in each camera separately, with redundant degrees of freedom. The inter-camera constraints are then enforced in a second stage. This late fusion from single- to multi-camera enables us to leverage the parallel performance provided by GPUs in the first data-intensive stage.

Computation times are mostly affected by the increased number of pixels for which the low-level vision cues need to be computed. Table I shows the required times (in ms) for a number of configurations. Each new frame needs to be copied to the GPU and participates three times in optical flow computation (image optical flow, tracker AR flow, detector AR flow) and thus also in Gabor filtering. Since the methods used are highly local and we use optimized implementations, we observe an approximately linear scaling in terms of pixels processed.

Since initially the object poses are estimated with redundant degrees of freedom, the number of *segments*, or objects effectively considered by the tracker, is the product of the actual number of objects and the number of cameras. Table II contains the computation times for one object (3 segments) and 40 objects (120 segments) when considering three cameras. Thanks to the late fusion approach, this increase hardly affects computation times. In each case we performed three internal iterations for robustness, and three external iterations to handle the non-linearity. The computation times required for composing the normal equations *is* affected by the number of measurements used. These measurements are the optical flow, AR flow, and depth measurements. We observe a sublinear scaling in Table II when increasing the number of measurements from 50,000 to 1,000,000. By allowing for so many measurements, the method can reduce the effects of imaging noise, modeling errors, and especially important in the multi-camera setting, inaccurate calibration.

In Table II we also show the times required to process a single full high-definition (1920×1080) image when considering the same number of segments. Note that there is an approximately fixed increase of around 6 ms. This is due to the increased number of sorting operations required for assigning measurements to segments. We use an efficient radix sort implementation with 8 bit radix [28].

To arrive at the total time required for tracking, we need to add an additional 5 ms mostly for rendering. The constrained update takes only 0.5 ms out of this for 3 cameras and 40 objects. So the total time for a configuration with three cameras, one million measurements, three internal and three external iterations and considering 40 objects equals 11.2 + 16.2 + 5.0 = 32.4 ms. This allows us to perform tracking at 30 Hz in this highly demanding configuration.

The SIFT-based pose detection runs on a separate GPU and provides pose estimates for the currently-selected camera and object at around 20 Hz for 640×480 resolution using SiftGPU [29]. It requires 20 ms for keypoint and descriptor computation, 20 ms for brute-force matching, and 10 ms for pnp pose estimation. Note that the detector does not slow down tracking since the pose detections are only used when available. Many improvements are possible in the detection component, such as faster feature extraction [30] and model size reduction [31], but we reserve these for future work.

B. Multiple Cameras Improve Tracking Accuracy

To measure the improvements that result from introducing additional cameras, we use simple and very accurate planar object models, see Fig. 3. We fix the three paper objects to the table so that they cannot move with respect to each other. Although they are essentially a single rigid object, for the purpose of accuracy evaluation, we let the tracker consider them as separate objects. As a consequence, relative differences can be observed in the pose estimates over time. This allows us to evaluate the tracker accuracy by measuring the deviation from a single rigid object hypothesis. Our error measure averages the frame-to-frame deviations from rigidity:

$$e_{\mathrm{nr}} = \sqrt{\frac{1}{f}\sum_t e_t} \ , \tag{22}$$

where f is the total number of frames, and:

$$e_t = \min_{\mathbf{T}_t^*} \frac{1}{n} \sum_i ||\mathbf{v}_{t+1}^i - \mathbf{T}_t^* \mathbf{v}_t^i||^2 \ , \tag{23}$$

represents the deviation from rigidity when transforming $\mathbf{v}_t$, the object vertices as hypothesized by the tracker's pose estimates at time t, to the object vertices hypothesized at the next frame, $\mathbf{v}_{t+1}$, according to the rigid transform $\mathbf{T}_t^*$ that minimizes the least-squares error. Here n is the total number of vertices in the model. The latter is found using a closed-form solution to the absolute orientation problem [32]. The results are summarized in Table III for two scenarios. The image sequences corresponding to these scenarios are part of the *supplemental material video*. In the first scenario, the table is moved by a human and only the robot occludes the scene. Note how the error increases in Table III as cameras are removed. The large error for the right arm camera is due to a severe occlusion of one of the objects by the robot. Figure 3 illustrates the accuracy difference between using only the left camera (panels A–D) and using all cameras (panels E–H). In the first case, a very precise alignment is obtained in the left camera (A,B) without guaranteeing consistency in the other views. Note especially how the purple object has shifted with respect to the others from C to D. Instead, when all cameras are used, the errors due to miscalibration are distributed across the cameras, and the large viewpoint differences resolve ambiguities and help maintain the relative position of the objects (G,H).

TABLE III
AVERAGE DEVIATION FROM RIGIDITY e_{nr} (IN MM)

	scenario	
active camera(s)	moving table	occlusion
head + left + right	0.33	0.51
left + right	0.39	1.02
head	0.59	2.98
left	0.57	4.70
right	6.16	3.27

In the second scenario a human occludes the objects. As can be expected we obtain a more dramatic increase in error when cameras are removed, see right column Table III. An example frame of this sequence is shown in Fig. 4A–C. See the *supplemental material video* for the complete sequence.

C. Detecting and Tracking Multiple Objects

We next show an example that illustrates how multiple moveable cameras can simplify the interpretation of complex cluttered scenes. The second column of Fig. 4D–F contains pose estimation results in a scenario with seven different objects. As the scene is highly cluttered and the objects are partially occluded in each viewpoint, this configuration can be resolved more easily by integrating the information from all viewpoints. Our method gradually detects and introduces the objects to the scene, while continuously refining their pose using the AR flow, so as to arrive at a globally consistent interpretation.

D. Object Stacking

In this final scenario we operate on a longer sequence where the PR2 uses both arms to stack a *Campbells* can on top of a cylindrical *Pringles* container. See Fig. 4G–L and the *supplemental material video* for the complete sequence. The execution has been entirely pre-programmed as we are only concerned with visual pose estimation in this work.

The problem is challenging because of the size of the can, the noise in the arm camera images, and the large distance between the head camera and the scene. The system is not accurately calibrated and this manifests itself in different ways in different configurations. Compare for example the

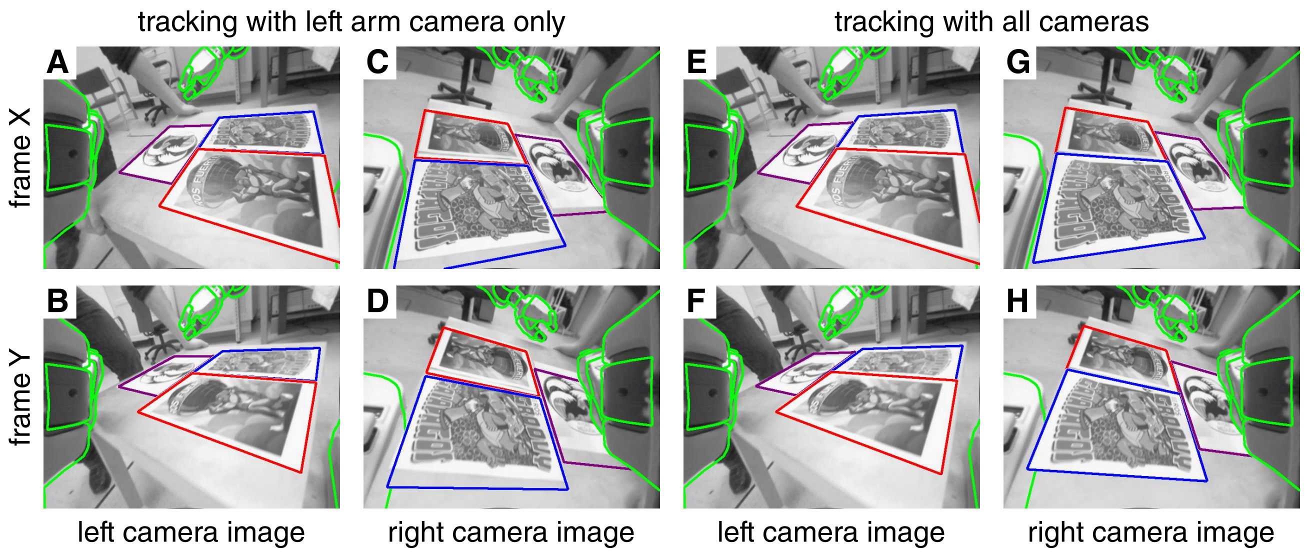

Fig. 3. Two frames (X and Y) from a scenario where a table is moved by a human. The three targets are rigidly attached to the table, but tracked independently. When tracking with the left camera only (A–D), high accuracy is obtained in that camera (A,B) but not in the right camera (C,D). The results are not only due to miscalibration since the purple target moves with respect to the others in C and D. Instead, when all cameras are used (E–H), the tracking is more robust and consistent in all cameras.

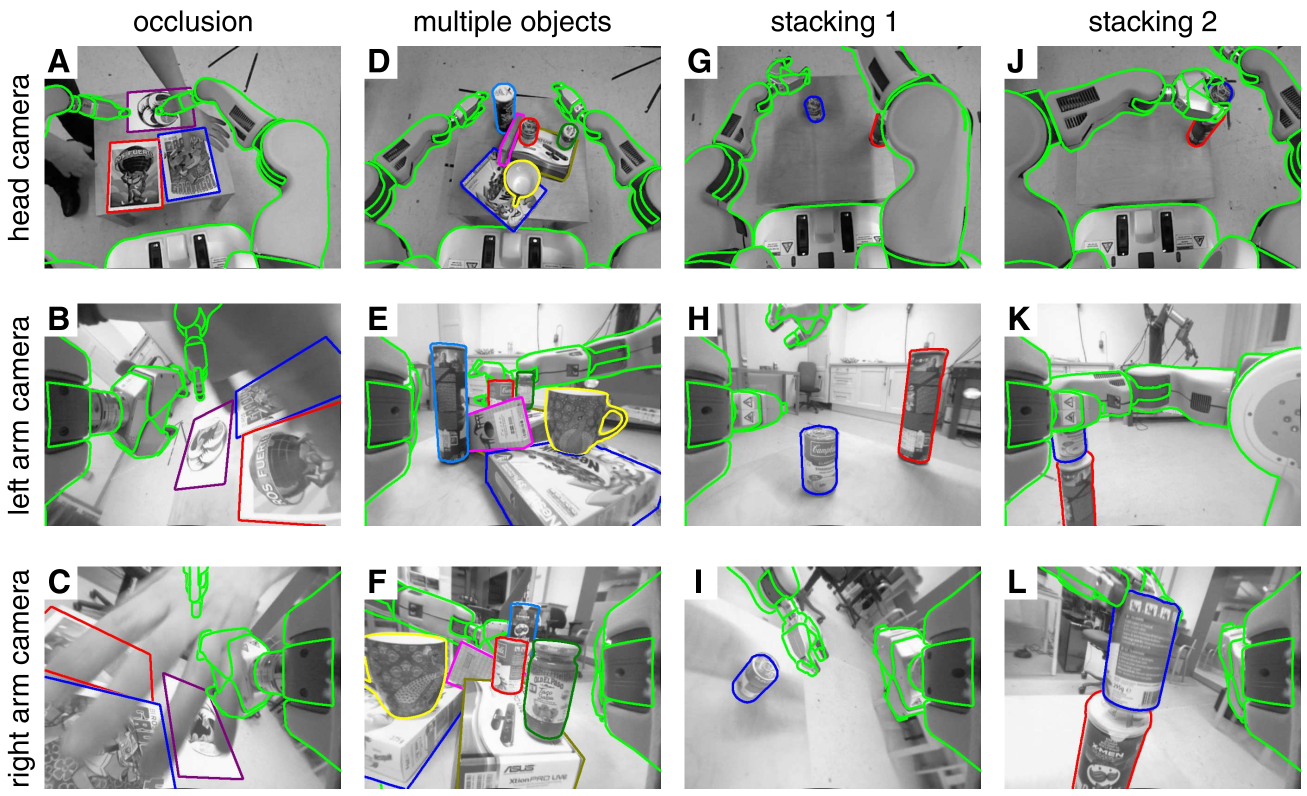

Fig. 4. The *occlusion*-scenario (A–C) demonstrates the robustness to unmodeled, external occlusions. The *multiple objects*-scenario (D–F) shows that the interaction between multi-camera tracking and detection can arrive at precise pose estimates in cluttered scenes where each viewpoint provides only partial visibility. The *stacking*-scenario (G–L) finally illustrates that our method maintains consistent object pose estimates in a complex stacking scenario even though the calibration is inaccurate, see the left arm as projected in the right arm camera frame.

different misalignment of the left gripper model in Figs. 4I and 4L.

Both arms need to collaborate to maintain full visibility of the scene. The small can is largely occluded by the left arm gripper but the right arm actively maintains a detailed view of the grasped object throughout the manipulation. At the same time the left arm camera maintains a view on the *Pringles* target, see Fig. 4H. The view from the head camera is frequently occluded by the manipulator and is too distant from the scene to provide useful SIFT features for 6DOF pose estimation. Collaboratively however, the three cameras can maintain very detailed information during all stages of the manipulation.

E. SimTrack Software Package

The proposed method is part of a larger ROS-based software framework for real-time pose detection and tracking. It can exploit monocular, stereo, and RGB-D cameras, and can easily incorporate different robots through their URDF-description. It consists of a tracker and detector component that can exploit multiple GPUs. The *SimTrack* package is released under a permissive BSD-license. We also provide detailed usage instructions, particularly in terms of how to obtain textured object meshes.

IV. Conclusion

We have proposed a robust object pose detection and tracking method that scales to multiple cameras and objects. We have demonstrated real-time performance in a complex configuration with three cameras and up to forty objects. We have shown that tracking accuracy improves when more cameras are exploited. In an extensive real-world stacking scenario, recorded using an imprecisely calibrated system, our method robustly detects and tracks the pose of all elements of interest. Since the main computational load in our approach involves data local to each camera, it allows for new robotic configurations with a large number of smart camera attachments. This opens up interesting research avenues concerned with optimal scene exploration for increased tracking accuracy, planning considering multi-camera visibility, on-line calibration, etc. The public release of our method allows for its immediate exploitation by the multiple PR2s and other robots used by the research community.

References

[1] D. Kragic and H. I. Christensen, "Survey on visual servoing for manipulation," Computational Vision and Active Perception Laboratory, Tech. Rep., 2002.

[2] O. Kermorgant and F. Chaumette, "Multi-sensor data fusion in sensor-based control: Application to multi-camera visual servoing," in *ICRA*, May 2011, pp. 4518–4523.

[3] F. Martin and R. Horaud, "Multiple-camera tracking of rigid objects," *Int. J. Robot. Res.*, vol. 21, no. 2, pp. 97–113, 2002.

[4] 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.

[5] I. Oikonomidis, M. Lourakis, and A. Argyros, "Evolutionary quasi-random search for hand articulations tracking," in *CVPR*, June 2014, pp. 3422–3429.

[6] K. Pauwels, V. Ivan, E. Ros, and S. Vijayakumar, "Real-time object pose recognition and tracking with an imprecisely calibrated moving RGB-D camera," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2014, pp. 2733–2740.

[7] 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.

[8] Z. Xie, A. Singh, J. Uang, K. Narayan, and P. Abbeel, "Multimodal blending for high-accuracy instance recognition," in *IROS*, Nov 2013, pp. 2214–2221.

[9] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze, "Multimodal cue integration through hypotheses verification for RGB-D object recognition and 6DOF pose estimation," in *ICRA*, 2013, pp. 2104–2111.

[10] A. Aldoma, T. Faulhammer, and M. Vincze, "Automation of 'ground truth' annotation for multi-view RGB-D object instance recognition datasets," in *IROS*, 2014, pp. 5016–5023.

[11] V. Pradeep, K. Konolige, and E. Berger, "Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach," in *Experimental Robotics*, O. Khatib, V. Kumar, and G. Sukhatme, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, vol. 79, pp. 211–225.

[12] N. T. Dantam, H. B. Amor, H. I. Christensen, and M. Stilman, "Online multi-camera registration for bimanual workspace trajectories," in *Humanoids*, 2014.

[13] H. Kato and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," in *2nd Int. Workshop on Augmented Reality (IWAR 99)*, Oct. 1999.

[14] T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, and M. Vincze, "BLORT - The blocks world robotic vision toolbox," in *ICRA*, 2010.

[15] E. Marchand, F. Spindler, and F. Chaumette, "ViSP for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robot. Autom. Mag.*, vol. 12, no. 4, pp. 40–52, Dec. 2005.

[16] Autodesk, "123D Catch," http://www.123dapp.com/catch/.

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

[18] "OGRE – Open Source 3D Graphics Engine," http://www.ogre3d.org.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," *ICRA Workshop on Open Source Software*, 2009.

[20] 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 *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Portland, 2013, pp. 2347–2354.

[21] K. Pauwels, M. Tomasi, J. Diaz Alonso, 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 Transactions on Computers*, vol. 61, no. 7, pp. 999–1012, July 2012.

[22] K. Pauwels, L. Rubio, and E. Ros, "Real-time model-based articulated object pose detection and tracking with variable rigidity constraints," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, Ohio, June 2014, pp. 3994–4001.

[23] C. Yang and G. Medioni, "Object modelling by registration of multiple range images," *Image Vision Comput.*, vol. 10, pp. 145–155, 1992.

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

[25] R. M. Murray and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

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

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

[28] NVIDIA, "CUB – CUDA unbound," http://nvlabs.github.io/cub.

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

[30] M. Björkman, N. Bergström, and D. Kragic, "Detecting, segmenting and tracking unknown objects using multi-label MRF inference," *Comput. Vis. Image Und.*, vol. 118, pp. 111–127, Jan. 2014.

[31] B. Leibe, A. Leonardis, and B. Schiele, "Robust object detection with interleaved categorization and segmentation," *Int. J. Comput. Vision*, vol. 77, no. 1-3, pp. 259–289, Nov. 2007.

[32] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, no. 4, pp. 629–642, 1987.