2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

RGB-D Object Tracking: A Particle Filter Approach on GPU

Changhyun Choi and Henrik I. Christensen
Center for Robotics & Intelligent Machines
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
{cchoi,hic}@cc.gatech.edu

***Abstract*— This paper presents a particle filtering approach for 6-DOF object pose tracking using an RGB-D camera. Our particle filter is massively parallelized in a modern GPU so that it exhibits real-time performance even with several thousand particles. Given an *a priori* 3D mesh model, the proposed approach renders the object model onto texture buffers in the GPU, and the rendered results are directly used by our parallelized likelihood evaluation. Both photometric (colors) and geometric (3D points and surface normals) features are employed to determine the likelihood of each particle with respect to a given RGB-D scene. Our approach is compared with a tracker in the PCL both quantitatively and qualitatively in synthetic and real RGB-D sequences, respectively.**

I. Introduction

As robots are getting gradually deployed away from structured environments to unstructured environments, the reliable interaction with the environments is a key necessity for the success of utilizing robotic systems. Especially, a robust and efficient object pose recognition is an important requirement for reliable robotic tasks. In early stages of robotic object perception, it was tried to solve this problem by employing known 3D object models [1]. The problem was typically formulated as estimating a pose that will best fit the given 3D model of the object to 2D image features: edges [2], [3] or line segments fitted from the edges [4]. The optimal pose or motion parameters were then estimated via efficient local optimizations.

Although edge-based tracking was shown to be usable for video rate tracking even in the early 90's [2], it was not robust to complex backgrounds and occlusions. Since edges themselves are not discriminative enough to provide reliable edge data associations, there have been many efforts to augment the early work by fusion with keypoint features [5], [6], [7] or maintaining multiple pose hypotheses on edge data associations [5], [8]. However, these approaches were still limited in the sense that they only considered a small number of pose hypotheses.

A more advanced formulation considering multiple pose hypotheses has appeared based on particle filtering. Particle filter [9], [10] is a Bayesian filtering method based on sequential simulation from posterior probability distributions. For the last two decades, this method has become popular since it can handle nonlinear and non-Gaussian dynamics, so it has often been regarded as a good alternative to the filtering methods designed upon Gaussian probability distributions for the state space. Isard and Blake [11] introduced a particle filter to the computer vision community as a robust 2D contour tracking approach. Since then, a large number of variants have been applied to the problem of body pose estimation [12], 3D object tracking [13], [14], [15], SLAM [16], etc.

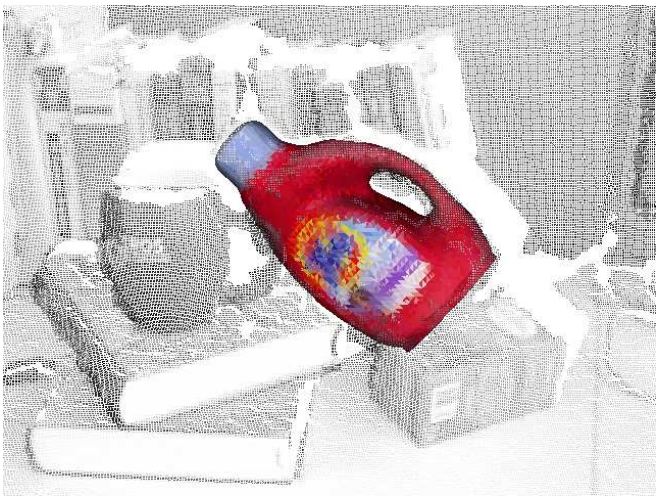

Fig. 1. **A tracking example.** An object is tracked via our particle filter parallelized on a GPU. The object rendering represents the mean of the particles.

However, adopting the particle filtering approaches to robotic perception has been limited mainly due to their high computational cost. To tackle this problem, some researchers [13], [14], [17], [18] have noticed that particle filter algorithms are inherently parallelizable. The main bottleneck of particle filters is in their likelihood evaluations. When N particles are employed to approximate the true posterior probability distribution, N independent and identical likelihood evaluations should be performed at each time step. Since one likelihood evaluation for a particle does not depend on the rest of the particles, the likelihood computation can be parallelized.

Modern graphics processing unit (GPU) provides massive parallel power beyond rendering purposes, so it is a natural idea to design a particle filter on the GPU for fast and robust object pose tracking. Montemayor et al. [17] showed a preliminary design of a particle filter on the GPU for a simple 2D object tracking. Klein and Murray [13] presented

a particle filter which tracks an arbitrarily shaped object in real-time. To maintain the high frame rate, the OpenGL occlusion query extension was employed to calculate the particle likelihood efficiently from a given edge image. While the work simply relied on an OpenGL extension, several approaches [14], [18] adopted a more advanced technique to exploit the GPU, NVIDIA CUDA™ technology, so that they render an object model to the GPU and a CUDA kernel directly accesses the rendering results, and then the importance weights of the particles are calculated from the differences between the rendered results and an input scene.

Most of the previous work have mainly used a monocular camera, so the visual features employed so far have been limited to edges [13], [14] or intensity differences [17], [18]. However, as RGB-D sensors were recently introduced, not only color but also depth information is readily available in real-time. This depth information enables us to use various geometric features, such as 3D point coordinates, surface normals, edges from depth discontinuities, high curvature regions, etc. Recently, this additional depth data has actively been used in many robotic and vision applications, such as human pose estimation [19], bin-picking problem [20], [21], [22], SLAM [23], [24], and object tracking [25]. The particle filter-based tracking in the PCL [26] is probably the closest system to our proposed solution. Given an object point cloud as a reference model, it can track the 6-DOF pose of the reference model over a sequence of RGB-D images. Its time consuming routines were parallelized in CPU, yet it does not achieve real-time performance with a sufficient number of particles (see Section VI). A brief description of the approach can be found in [27].

II. Contributions

We propose a robust particle filter parallelized on a GPU which can track a known 3D object model over a sequence of RGB-D images. Unlike the PCL tracking [27], we render the 3D object model to be used in the likelihood evaluation so that our approach can track the object in spite of significant pose variations. Our key contributions are as follows:

- We employ extended features to evaluate the likelihood of each particle state. While most of the previous work has mainly relied on 2D edges [13], [14] or intensity differences [17], [18] to calculate the importance weights of particles, we use both photometric (colors) and geometric features (3D points and surface normals) available from both RGB-D images and OpenGL rendering.
- We use the framebuffer object extension (FBO) in OpenGL and the CUDA OpenGL interoperability to reduce the mapping time of the rendering result to CUDA's memory space. As [14] mentioned, the mapping between OpenGL framebuffer and the memory space of CUDA takes as much as copying the rendered result to the memory space of CPU. To avoid this problem, our rendering is performed in the FBO so that the mapping time is nearly negligible.
- We devised an hierarchical approach to consider multiple renderings of the object. While the PCL tracking [26] maintains only one reference object cloud, we render the object to multiple viewports with different poses. It would be ideal if we draw all particle poses to the render buffers in the GPU, but it is not possible due to the memory limitation of the buffers. Instead, our approach renders the object to V viewports, and each particle searches the closest rendering from V viewports so that each likelihood evaluation is performed by transforming the closest rendered result with the current particle state (see Fig. 2).

To the best of our knowledge, our proposed solution is the first real-time particle filter for 6-DOF object pose tracking using rich visual features from the RGB-D sensor. Fig. 1 shows an example frame of our tracking where a target object is tracked in background clutter. The rendered 3D mesh model represents the mean of the particles for visualization purpose.

This paper is organized as follows. A particle filter for 6-DOF object pose tracking is briefly mentioned in Section III, and the likelihood function employing points, colors, and normals is introduced in Section IV. After the further explanation on the OpenGL and the CUDA implementation in Section V, our approach is compared with a baseline in both synthetic and real RGB-D image sequences in Section VI.

III. Particle Filter

In the 6-DOF pose tracking problem, a particle filter is employed to sample the trajectory of an object of interest over time. In a particle filtering framework, the posterior probability density function of the object pose $p(\mathbf{X}_t|\mathbf{Z}_{1:t})$ at the current time t is represented as a set of weighted particles by

$$\mathcal{S}_t = \{(\mathbf{X}_t^{(1)}, \pi_t^{(1)}), \ldots, (\mathbf{X}_t^{(N)}, \pi_t^{(N)})\} \tag{1}$$

where the particles $\mathbf{X}_t^{(n)} \in SE(3)$ represent samples, the normalized importance weights $\pi_t^{(n)}$ are proportional to the likelihood function $p(\mathbf{Z}_t|\mathbf{X}_t^{(n)})$, and N is the total number of the particles. At each time t, the particle filter performs the sequential importance sampling with resampling [10]. The current state $\mathcal{X}_t$ could be estimated by the weighted particle mean:

$$\mathcal{X}_t = \mathcal{E}[\mathcal{S}_t] = \sum_{n=1}^{N} \pi_t^{(n)} \mathbf{X}_t^{(n)}. \tag{2}$$

As we already showed in [15], when we estimate the mean, we need to obtain a valid rotation $\mathbf{R}_t \in SO(3)$ as the arithmetic mean $\overline{\mathbf{R}}_t = \frac{1}{N}\sum_{n=1}^{N} \mathbf{R}_t^{(n)}$ is not usually on the $SO(3)$ group. From [28], the desired mean rotation can be calculated via the orthogonal projection of the arithmetic mean as

$$\mathbf{R}_t = \begin{cases} \mathbf{V}\mathbf{U}^\mathsf{T} & \text{when } \det(\overline{\mathbf{R}}_t^\mathsf{T}) > 0 \\ \mathbf{V}\mathbf{H}\mathbf{U}^\mathsf{T} & \text{otherwise,} \end{cases} \tag{3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are estimated via the singular value decomposition of $\overline{\mathbf{R}}_t^\mathsf{T}$ (*i.e.* $\overline{\mathbf{R}}_t^\mathsf{T} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$) and $\mathbf{H} = \mathrm{diag}[1, 1, -1]$.

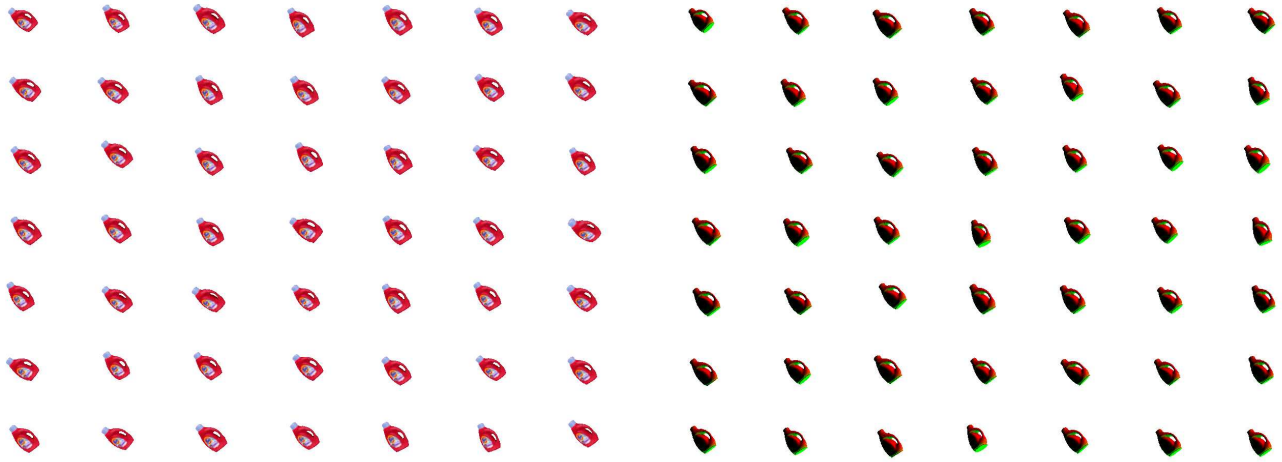

Fig. 2. **Multiple renderings for the likelihood evaluation.** The object of interest is rendered with the first V particle states and the likelihoods of the V particles are evaluated from the rendering results. For the rest particles, each particle finds the closest rendering and use the rendering result to evaluate its likelihood. Left and right images represent the color and normal renderings in the GPU. (Best viewed in color)

IV. Likelihood Evaluation

Designing an efficient and robust likelihood function is crucial, since it directly determines the overall performance of the particle filtering in terms of both time and accuracy. When an RGB-D camera is considered, there are various measurements we can employ: 3D point coordinates, colors of points, surface normals, curvature, edges from depth discontinuities or surface textures, etc. In this work, we choose the point coordinates $\mathbf{x} \in \mathbb{R}^3$ and their associated colors $\mathbf{c} \in \mathbb{R}^3$ and normals $\mathbf{n} \in \mathbb{R}^3$. Thus, a measurement point $\mathbf{p}$ is defined as

$$\mathbf{p} = (\mathbf{x}^\mathsf{T}, \mathbf{n}^\mathsf{T}, \mathbf{c}^\mathsf{T})^\mathsf{T} \in \mathbb{R}^9. \tag{4}$$

For clear notation, let us define accessing operators for $\mathbf{p}$ such that

$$\mathrm{x}(\mathbf{p}) = (\mathbf{x}^\mathsf{T}\ 1)^\mathsf{T} \in \mathbb{R}^4 \tag{5}$$

$$\mathrm{n}(\mathbf{p}) = (\mathbf{n}^\mathsf{T}\ 1)^\mathsf{T} \in \mathbb{R}^4 \tag{6}$$

$$\mathrm{c}(\mathbf{p}) = \mathbf{c} \in \mathbb{R}^3. \tag{7}$$

The reason we choose the three measurements is that this combination allows us to perform direct comparisons between the given RGB-D scene and the rendering results from the computer graphics pipeline. Hence, we can efficiently calculate the likelihood for a large number of particles.

Given the current pose hypothesis $\mathbf{X}_t^{(n)}$ and the rendered object model $\mathbf{M}_t$, the likelihood of the scene $\mathbf{Z}_t$ is defined as

$$p(\mathbf{Z}_t|\mathbf{X}_t^{(n)}, \mathbf{M}_t) = \prod_{(i,j)\in\mathcal{A}} p(\mathbf{z}_t^{(i)}|\mathbf{X}_t^{(n)}, \mathbf{m}_t^{(j)}) \tag{8}$$

where $\mathcal{A} = \{(i,j)|\ \mathrm{proj}(\mathrm{x}(\mathbf{z}_t^{(i)})) = \mathrm{proj}(\mathbf{X}_t^{(n)} \cdot \mathrm{x}(\mathbf{m}_t^{(j)}))\}$ is the set of point associations between the scene $\mathbf{Z}_t$ and the object model $\mathbf{M}_t$, and $\mathbf{z}_t^{(i)}, \mathbf{m}_t^{(j)} \in \mathbb{R}^9$ are corresponding points in the scene and model, respectively. The operator $\mathrm{proj}(\cdot)$ calculates 2D image coordinates of given 3D homogeneous point coordinates by projecting the point with the known camera intrinsic parameters $\mathbf{K} \in \mathbb{R}^{3\times3}$. With the proj operator, the point associations $\mathcal{A}$ can be efficiently determined. The likelihood of each association (i,j) is then defined as

$$\begin{aligned} p(\mathbf{z}_t^{(i)}|\mathbf{X}_t^{(n)}, \mathbf{m}_t^{(j)}) = &\exp^{-\lambda_e \cdot d_e(\mathrm{x}(\mathbf{z}_t^{(i)}),\ \mathbf{X}_t^{(n)}\cdot\mathrm{x}(\mathbf{m}_t^{(j)}))} \\ &\cdot \exp^{-\lambda_n \cdot d_n(\mathrm{n}(\mathbf{z}_t^{(i)}),\ \mathbf{X}_t^{(n)}\cdot\mathrm{n}(\mathbf{m}_t^{(j)}))} \\ &\cdot \exp^{-\lambda_c \cdot d_c(\mathrm{c}(\mathbf{z}_t^{(i)}),\ \mathrm{c}(\mathbf{m}_t^{(j)}))} \end{aligned} \tag{9}$$

where $d_e(\mathbf{x}_1, \mathbf{x}_2)$, $d_n(\mathbf{n}_1, \mathbf{n}_2)$, and $d_c(\mathbf{c}_1, \mathbf{c}_2)$ are Euclidean, normal, and color distances as shown below

$$d_e(\mathbf{x}_1, \mathbf{x}_2) = \begin{cases} \|\mathbf{x}_1 - \mathbf{x}_2\| & \text{if } \|\mathbf{x}_1 - \mathbf{x}_2\| \le \tau \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

$$d_n(\mathbf{n}_1, \mathbf{n}_2) = \frac{\cos^{-1}(\mathbf{n}_1^\mathsf{T}\mathbf{n}_2 - 1)}{\pi} \tag{11}$$

$$d_c(\mathbf{c}_1, \mathbf{c}_2) = \|\mathbf{c}_1 - \mathbf{c}_2\| \tag{12}$$

and $\lambda_e, \lambda_n, \lambda_c$ are the parameters that determines the sensitivity of the distances to the likelihood. The τ in (10) is a threshold value for the Euclidean distance between the two points. Note that $\mathbf{n}_1, \mathbf{n}_2 \in \mathbb{R}^4$ in (11) are homogeneous point coordinates, so 1 need to be subtracted from the inner product. For the color distance in (12), any kind of color space can be considered as long as $0 \le d_c(\mathbf{c}_1, \mathbf{c}_2) \le 1$, but we adopted the HSV color space due mainly to its invariance to illumination changes. Please note that the point and normal coordinates of object model point $\mathbf{m}_t^{(j)}$ are in the object coordinate frame, so the transformed point by the current pose $\mathbf{X}_t^{(n)}$ should be considered to calculate the distances.

V. Implementation Details

As we already mentioned in Section II, we render our object of interest onto V viewports in the render buffers with the first V particle poses. For the rest of the particles, each particle finds a closest rendering with respect to the pose hypothesis and transforms the closet rendering result with the current pose. For this calculation, we need to access the color,

Fig. 3. **Mesh models for objects and kitchen.** Object models were generated by fusing multiple RGB-D views, followed by running the Poisson reconstruction algorithm [29]. To generate a set of synthetic sequences, a kitchen model was download from the Google 3D warehouse. From left to right, 'Tide', 'Milk', 'Orange Juice', 'Kinect Box', and 'Kitchen'.

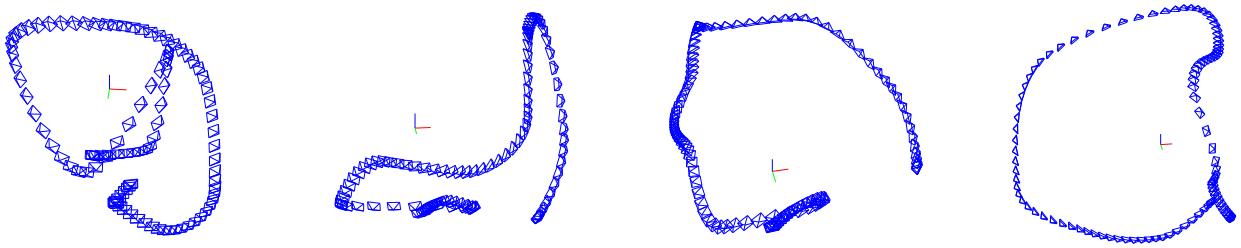

Fig. 4. **Camera trajectories in synthetic sequences.** Synthetic RGB-D sequences were generated with their corresponding ground truth trajectories of the objects. Please note the significant variations in translation, rotation, and velocity. From left to right, trajectories of 'Tide', 'Milk', 'Orange Juice', and 'Kinect Box'.

vertex (*i.e.* 3D point), and normal information for all visible points in the rendered result. It is relatively straightforward to get color information via accessing the render buffer, but it is tricky to access 3D point and 3D normal data from the buffer. To tackle this problem, we employ the OpenGL Shading Language (GLSL) to directly access the point and normal data in the middle of the graphics pipeline. For point information, we designed a set of simple vertex and fragment shaders so that the 3D coordinates of visible points `gl_Vertex` are saved to color texture points `gl_FragColor`. Similarly, for normal data another set of vertex and fragment shaders is used so that the surface normals of the object `gl_Normal` are saved in color texture points. Note that these points and normals are in the object coordinate frame, so we do not need to perform an inverse transform on the rendering result before transforming them with respect to the current particle pose.

The main purpose of the multiple viewports rendering is not for visualization but for the likelihood evaluation of each particle. Thus using the Frame Butter Object (FBO, `GL_ARB_framebuffer_object`) is a good choice for our rendering purpose. The FBO is an OpenGL extension for off-screen framebuffers. Unlike the default framebuffer of OpenGL provided by window systems, the FBO is more flexible and efficient since all resources bound in the FBO are shared within the same context. The FBO allows users to attach multiple texture images to color attachments. For our purpose, we attach three texture images to the three color attachments: `GL_COLOR_ATTACHMENT0` for color data, `GL_COLOR_ATTACHMENT1` for point data, and `GL_COLOR_ATTACHMENT2` for normal data. In the rendering phase, our object of interest is drawn onto each color attachment. While the color texture is drawn using the usual OpenGL rendering, the point and normal textures are rendered via the aforementioned shader programs. For each rendering, the object is rendered to V viewports by calling `glViewport()`.

After rendering the object with shader programs, we evaluate the likelihood function on the GPU. To utilize the texture images attached to the FBO in our likelihood evaluation kernel, we use the CUDA OpenGL interoperability that allows to map/unmap OpenGL buffers to CUDA's memory space. So our CUDA kernel can access the rendered buffers very efficiently.

VI. Experiments

In this section, we present a set of comparative experiments between the PCL tracking and our proposed approach. The performance of the two approaches is quantitatively evaluated using a set of synthetic RGB-D sequences with the ground truth object trajectories in Section VI-B, followed by the qualitative evaluation using real RGB-D sequences in Section VI-C. For the evaluations, both trackers are initialized with the known ground truth in synthetic sequences and with the converged pose estimates after running our tracker from a sufficiently close initial pose. The PCL tracking provides an option for adaptive particle size based on [30], but here the fixed particle size is considered for fair comparisons with our approach and for the performance evaluation with respect to different particle sizes. All experiments were performed using a standard desktop computer (Intel Core2 Quad CPU Q9300, 8G RAM) with an off-the-shelf GPU

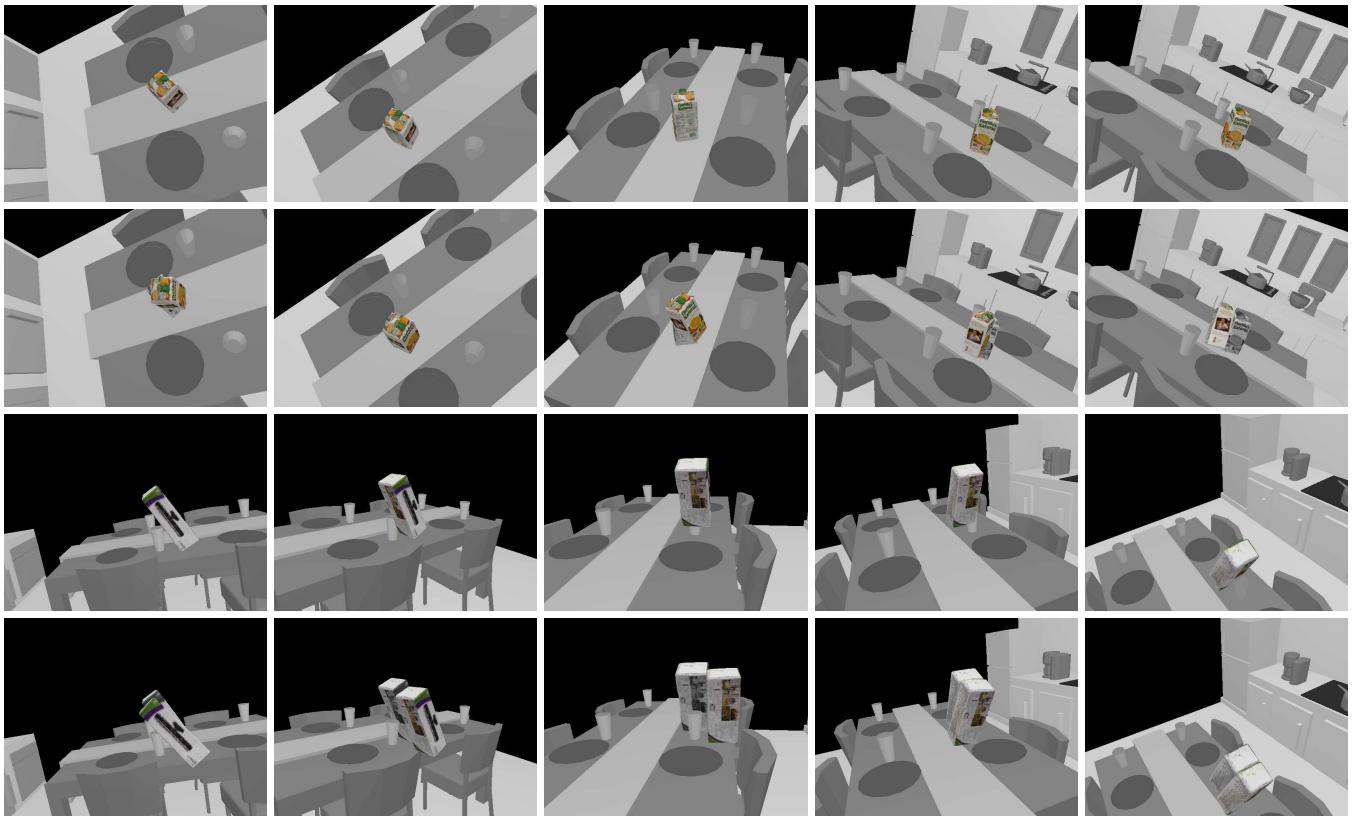

Fig. 5. **Tracking results on the 'Orange Juice' and the 'Kinect Box' synthetic sequences.** For both sequences, the upper rows show the tracking results of our approach, while the lower rows present the results of the PCL tracking ($N = 6400$ for the 'Orange Juice' sequence and $N = 12800$ for the 'Kinect Box' sequence). In both sequences, our approach tracks the true object trajectories well, but the PCL tracking is often lost due to the limitation of the object model. (Best viewed in color)

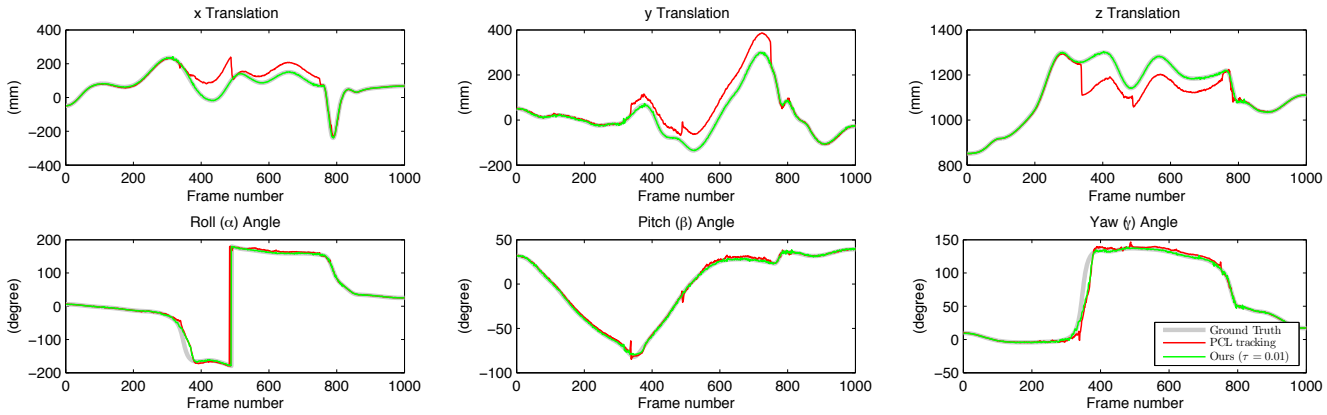

Fig. 6. **The 6-DOF pose plots of the 'Kinect Box' results in Fig. 5.** While our approach well follows the ground truth, the PCL tracking suffers from wrong tracking due to the limitation of the object model.

(NVIDIA GeForce GTX 590, CUDA 4.1) and an RGB-D camera (ASUS Xtion Pro Live).

A. Object Models

For the experiments, four objects were chosen, and the 3D mesh models of the objects (Fig. 3) were generated by using an RGB-D sensor. To generate the mesh model, we first obtained multiple RGB-D views and registered them. We could use one of the RGB-D SLAM approaches [23], [24], [31] to register the multiple views, but we employed several ARTags for simplicity. Once the multiple views were fused, the object cloud was segmented from the background clouds and was reconstructed to result in a mesh model via the Poisson reconstruction algorithm [29]. As the PCL tracking can not use the 3D mesh models, the object point clouds which were obtained by rendering with the initial poses were fed to be used as reference models.

B. Synthetic Sequences

For a quantitative analyses, we generated a set of synthetic RGB-D sequences. To simulate a realistic environment, a virtual kitchen model (see Fig. 3) was downloaded from the

TABLE I

RMS ERRORS AND COMPUTATION TIME IN SYNTHETIC RGB-D SEQUENCES (PCL VS. OUR TRACKING)

Objects	Tracker	N	RMS Errors†						Time (ms)†
			X (mm)	**Y** (mm)	**Z** (mm)	**Roll** (deg)	**Pitch** (deg)	**Yaw** (deg)	
Tide	PCL	100	**3.43**	**4.96**	**3.09**	9.10	**4.05**	5.85	81.46
		200	**2.45**	**3.66**	**2.38**	**6.96**	**2.88**	**4.34**	106.50
		400	**2.13**	**3.00**	**1.89**	**5.85**	**2.53**	3.50	152.57
		800	**1.74**	**2.79**	**1.59**	5.43	2.35	3.20	253.33
		1600	1.69	2.61	**1.51**	5.58	2.23	3.39	430.15
		3200	1.49	2.50	**1.11**	5.26	2.17	3.00	744.50
		6400	1.34	2.11	**0.95**	5.02	2.15	2.93	1376.92
		12800	1.46	2.25	**0.92**	5.15	2.13	2.98	2762.73
	Ours ($\tau = 0.01$)	100	3.47	6.55	4.30	**7.58**	5.14	**4.96**	**41.48**
		200	3.51	4.16	2.71	8.17	3.12	4.55	**41.55**
		400	2.33	3.25	2.53	5.19	2.26	**2.86**	**42.00**
		800	1.88	2.96	2.24	**3.79**	**1.93**	**2.32**	**42.37**
		1600	**1.66**	**2.42**	1.91	**3.48**	**1.71**	**2.21**	**44.60**
		3200	**1.27**	**1.87**	1.54	**2.43**	**1.36**	**1.58**	**48.93**
		6400	**1.14**	**1.54**	1.42	**2.25**	**1.13**	**1.39**	**77.88**
		12800	**0.83**	**1.37**	1.20	**1.78**	**1.09**	**1.13**	**111.48**
Milk	PCL	100	**3.04**	7.91	3.60	62.43	37.62	50.76	74.09
		200	2.48	6.65	3.33	62.37	37.76	50.82	91.37
		400	2.37	4.89	2.35	50.49	33.71	41.33	130.47
		800	2.36	4.81	2.03	52.23	33.78	42.89	208.83
		1600	1.78	3.99	**1.69**	49.27	33.65	40.22	351.83
		3200	13.68	43.72	24.92	64.45	21.52	74.78	585.79
		6400	2.03	4.24	1.57	55.55	34.51	45.67	1126.65
		12800	13.38	31.45	26.09	59.37	19.58	75.03	2205.18
	Ours ($\tau = 0.01$)	100	3.51	**5.96**	**2.95**	**12.18**	**3.36**	**11.19**	**49.14**
		200	**2.42**	**4.79**	**2.52**	**14.06**	**2.97**	**12.55**	**50.21**
		400	**1.99**	**4.00**	**2.21**	**10.50**	**2.57**	**9.29**	**50.28**
		800	**1.79**	**3.16**	**1.97**	**7.77**	**2.03**	**6.96**	**51.32**
		1600	**1.28**	**2.55**	1.74	**4.35**	**1.68**	**3.90**	**52.95**
		3200	**1.20**	**2.37**	**1.41**	**6.22**	**1.74**	**5.49**	**60.55**
		6400	**1.05**	**2.07**	**1.21**	**4.00**	**1.44**	**3.47**	**94.55**
		12800	**0.93**	**1.94**	**1.09**	**3.83**	**1.41**	**3.26**	**133.95**
Orange Juice	PCL	100	4.83	4.95	3.18	84.38	41.50	46.51	74.15
		200	3.58	**3.47**	2.99	84.12	44.11	44.76	88.67
		400	3.21	**2.83**	2.49	86.48	44.67	45.26	114.36
		800	3.06	2.54	2.44	84.76	42.36	45.87	182.19
		1600	2.61	2.39	2.11	84.42	41.65	46.37	295.26
		3200	26.76	5.18	10.83	97.34	51.81	50.67	487.15
		6400	26.86	5.30	11.19	92.67	53.43	65.90	896.85
		12800	2.53	2.20	1.91	85.81	42.12	46.37	1637.13
	Ours ($\tau = 0.01$)	100	**3.49**	**4.19**	**2.70**	**6.82**	**2.86**	**6.55**	**50.39**
		200	**3.39**	4.05	**2.41**	**5.02**	**2.23**	**5.81**	**50.19**
		400	**2.86**	3.64	**2.14**	**3.88**	**1.77**	**4.68**	**50.54**
		800	**2.18**	**2.53**	**1.94**	**2.55**	**1.44**	**2.80**	**50.64**
		1600	**1.86**	2.39	**1.79**	**2.36**	**1.29**	**2.76**	**52.58**
		3200	**1.56**	**2.17**	**1.50**	**1.54**	**1.09**	**2.09**	**56.74**
		6400	**1.12**	**1.61**	**1.45**	**1.54**	**0.84**	**1.61**	**80.22**
		12800	**0.96**	**1.44**	**1.17**	**1.32**	**0.75**	**1.39**	**117.08**
Kinect Box	PCL	100	50.02	48.08	54.28	**12.78**	**2.98**	19.77	96.23
		200	44.45	71.22	53.61	86.93	34.92	64.54	146.30
		400	32.62	51.28	51.01	12.46	2.43	12.56	238.51
		800	44.83	43.50	56.51	10.23	2.31	10.63	395.86
		1600	43.93	42.70	55.70	9.80	2.33	10.58	666.85
		3200	44.59	42.93	55.78	11.82	1.94	11.33	1218.44
		6400	43.43	41.92	55.78	**7.21**	2.08	7.74	2377.42
		12800	43.99	42.51	55.89	7.62	1.87	8.31	4539.42
	Ours ($\tau = 0.01$)	100	**11.26**	**30.02**	**19.66**	15.86	3.75	**14.51**	**50.07**
		200	**7.67**	**18.51**	**12.02**	**11.27**	**2.18**	**11.15**	**49.74**
		400	**5.61**	**13.86**	**9.24**	**10.12**	**1.71**	**10.16**	**52.21**
		800	**3.67**	**5.28**	**2.60**	**8.40**	**1.58**	**8.43**	**51.87**
		1600	**3.37**	**4.13**	**2.15**	**7.86**	**1.17**	**7.86**	**56.33**
		3200	**6.11**	**9.16**	**7.51**	**7.63**	**1.03**	**7.51**	**64.00**
		6400	**2.38**	**3.28**	**1.52**	8.14	**1.42**	**7.52**	**116.17**
		12800	**1.84**	**2.23**	**1.36**	**6.41**	**0.76**	**6.32**	**166.14**

† For the sake of comparison, better results are indicated in bold type.

Fig. 7. **Tracking results on the 'Tide' and 'Milk' real sequences.** For both sequences, the upper rows show the tracking results of our approach, while the lower rows present the results of the PCL tracking ($N = 1600$). As the objects undergo significant variations in rotations, the PCL tracking suffered from false pose estimates. Thanks to multiple model renderings every time, our tracking reliably tracks the true poses of the objects. (Best viewed in color)

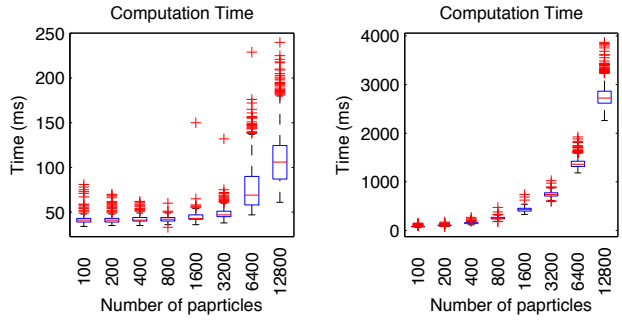

Fig. 8. **Boxplots showing computation time of our tracking (left) and the PCL tracking (right) on the 'Tide' synthetic sequence.** For both approaches, time linearly increases as the number of the particles N increases, but our increasing rate is much smaller. Note that the computation time of our approach is less than 50 ms until $N \leq 3200$.

Google 3D warehouse. After placing each object model in the kitchen model, a set of synthetic sequences was generated by moving the virtual camera around the object. Fig. 4 shows the camera trajectories of the synthetic data which exhibit high variations in translation, rotation, and velocity of the camera. The object trajectories were saved to be used as ground truth poses of the objects with respect to the camera coordinate frame.

To compare our approach with the PCL tracking, we calculated the root mean square (RMS) errors and average computation time per frame over the four synthetic RGB-D sequences as shown in Table I. For the sake of comparison, better results are indicated in bold type. The RMS errors vary depending on both the object type and the difficulty of the sequences. But the overall trend here is that as the number of the particles N increases the translational and rotational errors are decreased. In the 'Tide' sequence which is rather simple compared to the other sequences, for example, the PCL tracking reports slightly better results when $N \leq 800$. But as N increases, our tracker shows more accurate results. An interesting fact in the 'Tide' sequence is that the PCL tracking shows slightly better result in $\mathbf{z}$ translation. This may be due to the fact that the PCL tracking has only one reference object point cloud as an object model so that it does result in smaller error in that direction. However, the limited number of the reference cloud is getting problematic when it runs on more challenging sequences. Please note that big errors in both translation and rotations in the 'Milk', 'Orange Juice', and 'Kinect Box' sequences. Since the objects are self-symmetric themselves, the one reference view of each object is not enough to track the objects over the entire sequences. As shown in Fig. 5 and Fig. 6, the PCL tracking is often stuck in local minima during the tracking, while our approach robustly tracks the objects in the sequences.

For robotic applications, the computation time is really important since it directly determines the performance and the reliability of the robotic systems. As we can see in both Table I and Fig. 8, the computation time of both approaches

increases linearly as N increases. However, our tracking only takes about 50 ms per frame (*i.e.* 20 frames per second) with several thousands particles, whereas the PCL tracking suffers from low frame rates. Although the PCL tracking shows comparable performance in the 'Tide' sequence, if we consider the real frame rates the PCL tracking would have much higher errors due to the lost frames.

C. Real Sequences

We ran both tracking approaches in a set of real image sequences which exhibits significant noise and more variable object motions compared to the synthetic sequences. Fig. 7 shows the pose tracking results on the 'Tide' and 'Milk' sequences. For clear visualization, the RGB channels from the RGB-D sequences were converted to gray scale, and each rendered object model was drawn on top of them. Invalid depth points (also known as Nan points) were shown as black points. For both sequences, $N = 1600$ particles were employed. As the objects experience significant variations in rotations, the PCL tracking (lower rows in each sequence) often loses its tracking. Thanks to employing the 3D object model and rendering it in multiple views, our tracking dependably tracks the object in spite of the challenging rotational motions.

VII. Conclusions

We presented an approach for RGB-D object tracking which is based on a particle filter on a GPU. Rich visual features were employed to evaluate the likelihood of each particle, and this process, which is a typical bottleneck in most particle filters, was parallelized on the GPU so that our proposed solution achieves real-time performance. Through a set of extensive experiments with both synthetic and real RGB-D sequences, we verified that our approach is not only faster but also more accurate than the PCL tracking.

VIII. Acknowledgments

This work has in part been sponsored by the Boeing Corporation. The support is gratefully acknowledged.

References

[1] L. G. Roberts, "Machine perception of three-dimensional solids," Ph.D. dissertation, MIT Press, 1965.

[2] C. Harris and C. Stennett, "RAPID-a video rate object tracker," in *Proc. British Machine Vision Conf. (BMVC)*, 1990, pp. 73–77.

[3] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 932–946, 2002.

[4] D. G. Lowe, "Fitting parameterized three-dimensional models to images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 5, pp. 441–450, 2002.

[5] L. Vacchetti, V. Lepetit, and P. Fua, "Combining edge and texture information for real-time accurate 3D camera tracking," in *Proc. Int'l Symposium on Mixed and Augmented Reality (ISMAR)*, 2004.

[6] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Proc. IEEE Int'l Conf. Computer Vision (ICCV)*, vol. 2, 2005.

[7] C. Choi and H. I. Christensen, "Real-time 3D model-based tracking using edge and keypoint features for robotic manipulation," in *Proc. IEEE Int'l Conf. Robotics Automation (ICRA)*, 2010, pp. 4048–4055.

[8] C. Kemp and T. Drummond, "Dynamic measurement clustering to aid real time tracking," in *Proc. IEEE Int'l Conf. Computer Vision (ICCV)*, 2005, pp. 1500–1507.

[9] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proceedings F Radar and Signal Processing*, vol. 140, 1993, pp. 107–113.

[10] A. Doucet, N. De Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice*. Springer New York, 2001, vol. 1.

[11] M. Isard and A. Blake, "Condensation–conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

[12] J. Deutscher, A. Blake, and I. Reid, "Articulated body motion capture by annealed particle filtering," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2000, pp. 126–133 vol.2.

[13] G. Klein and D. Murray, "Full-3D edge tracking with a particle filter," in *Proc. British Machine Vision Conf. (BMVC)*, 2006.

[14] P. Azad, D. Munch, T. Asfour, and R. Dillmann, "6-DoF model-based tracking of arbitrarily shaped 3D objects," in *Proc. IEEE Int'l Conf. Robotics Automation (ICRA)*, 2011, pp. 5204–5209.

[15] C. Choi and H. I. Christensen, "Robust 3D visual tracking using particle filtering on the special euclidean group: A combined approach of keypoint and edge features," *International Journal of Robotics Research*, vol. 31, no. 4, pp. 498–519, Apr. 2012.

[16] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI)*, vol. 18, 2003, pp. 1151–1156.

[17] A. S. Montemayor, J. J. Pantrigo, n. Snchez, and F. Fernndez, "Particle filter on GPUs for real-time tracking," in *ACM SIGGRAPH 2004 Posters*, 2004, p. 94.

[18] O. Mateo Lozano and K. Otsuka, "Real-time visual tracker by stream processing," *Journal of Signal Processing Systems*, vol. 57, no. 2, pp. 285–295, 2009.

[19] 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 Conf. Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2011, p. 7.

[20] M. Germann, M. D. Breitenstein, H. Pfister, and I. K. Park, "Automatic pose estimation for range images on the GPU," in *Proc. Int'l Conf. 3-D Digital Imaging and Modeling (3DIM)*, 2007, pp. 81–90.

[21] C. Choi, Y. Taguchi, O. Tuzel, M.-Y. Liu, and S. Ramalingam, "Voting-based pose estimation for robotic assembly using a 3D sensor," in *Proc. IEEE Int'l Conf. Robotics Automation (ICRA)*, 2012.

[22] C. Choi and H. Christensen, "3D pose estimation of daily objects using an RGB-D camera," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots Systems (IROS)*, Oct., pp. 3342–3349.

[23] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," in *Proc. Int'l Symposium on Experimental Robotics (ISER)*, 2010.

[24] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Proc. Int'l Symposium on Mixed and Augmented Reality (ISMAR)*, 2011, pp. 127–136.

[25] Y. Park, V. Lepetit, and W. Woo, "Texture-less object tracking with online training using an RGB-D camera," in *Proc. Int'l Symposium on Mixed and Augmented Reality (ISMAR)*, 2011, pp. 121–126.

[26] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proc. IEEE Int'l Conf. Robotics Automation (ICRA)*, 2011, pp. 1–4.

[27] C. Bersch, D. Pangercic, S. Osentoski, K. Hausman, Z.-C. Marton, R. Ueda, K. Okada, and M. Beetz, "Segmentation of textured and textureless objects through interactive perception," in *RSS Workshop on Robots in Clutter: Manipulation, Perception and Navigation in Human Environments*, 2012.

[28] M. Moakher, "Means and averaging in the group of rotations," *SIAM Journal on Matrix Analysis and Applications*, vol. 24, no. 1, pp. 1–16, 2003.

[29] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of Eurographics Symposium on Geometry processing*, 2006.

[30] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *International Journal of Robotics Research*, vol. 22, no. 12, pp. 985–1003, 2003.

[31] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots Systems (IROS)*, 2012.