Robot Arm Pose Estimation through Pixel-Wise Part Classification

Jeannette Bohg[1], Javier Romero[2], Alexander Herzog[1] and Stefan Schaal[1,3]

***Abstract*— We propose to frame the problem of *marker-less* robot arm pose estimation as a pixel-wise part classification problem. As input, we use a depth image in which each pixel is classified to be either from a particular robot part or the background. The classifier is a random decision forest trained on a large number of synthetically generated and labeled depth images. From all the training samples ending up at a leaf node, a set of offsets is learned that votes for relative joint positions. Pooling these votes over all foreground pixels and subsequent clustering gives us an estimate of the true joint positions. Due to the intrinsic parallelism of pixel-wise classification, this approach can run in super real-time and is more efficient than previous ICP-like methods. We quantitatively evaluate the accuracy of this approach on synthetic data. We also demonstrate that the method produces accurate joint estimates on real data despite being purely trained on synthetic data.**

I. INTRODUCTION

Autonomous manipulation and grasping of objects is one of the remaining key challenges within robotics. Tremendous progress has been made in the area of data-driven grasp synthesis in recent years. An extensive review of existing approaches can be found in [2]. The majority of the current methods tries to visually infer a grasp configuration which is then executed in an open-loop manner. However, this decoupled perception-action scheme can lead to a poor success rate in a real-world situation which may involve sensor noise, inaccurate system models, state uncertainty or a dynamically changing environment. There has been some work on closed-loop execution where the current grasp configuration is adapted based on either visual or haptic sensory feedback, as for example in [26, 12, 13, 16]. This approach has proven to be more robust against the aforementioned factors.

In this paper, we are interested in improving grasp quality by exploiting visual feedback in a continuous, dense and fast manner. This contrasts with tactile-sensor feedback, which is valuable only at prehension time, and with feature or edge-based visual feedback, which may be sparse, non-robust and sometimes slow. Continuous perception of a grasping task as for example required by techniques such as visual servoing [7] involves the simultaneous tracking of object and end-effector. We have previously addressed the problem of object tracking during manipulation, where the main challenge is to cope with strong occlusions by either the robotic or human hand [27]. For the remaining task of arm tracking, joint encoder readings can be exploited to compute the arm position relative to the camera. However, this information can be erroneous due to calibration inaccuracies, sensor noise or external forces. Moreover, under-actuated manipulators may altogether lack such information. Therefore in many robotic systems, there exists a gap between pro-prioception (where the robot *thinks* its arm is) and reality as for example visualized in Fig. 1. Especially for fine manipulation tasks, this becomes a big hurdle for grasp success.

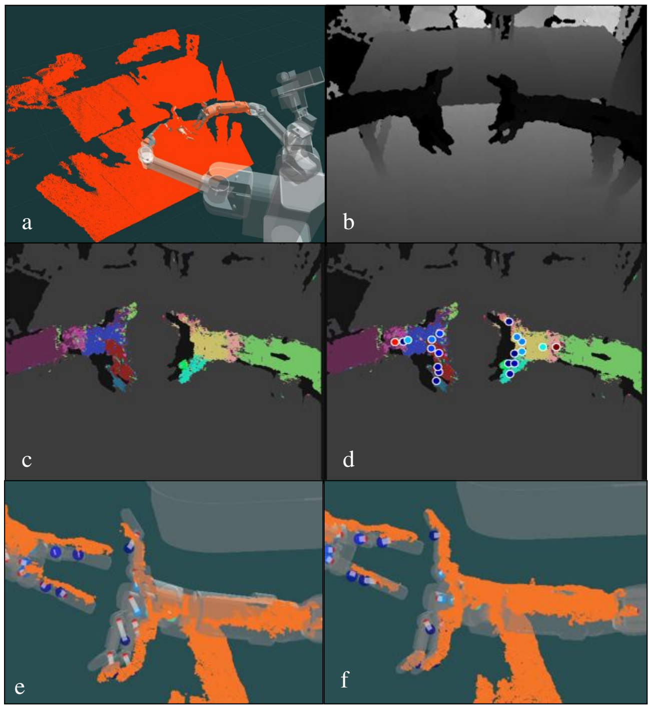


Fig. 1: General overview of the system. The system minimizes the forward kinematic error based on a single depth image from an Xtion depth sensor. (a) ARM robot [19] observing a real scene with its depth sensor. The encoder information from the robot is inaccurate (see (e)). (b) Depth image from Xtion. (c) Automatic per-pixel segmentation of the depth image into robot parts and background. (d) Estimation of the 3D joint positions and back-projected into the image. (e) Encoder estimated positions (small red spheres) and visually estimated ones (larger colored spheres), together with the real point cloud (orange) and the (erroneous) robot model as inferred from the joint encoders. The offset between point cloud and 3D model is due to calibration inaccuracies and encoder errors. Estimated joint positions are linked to their encoder counterparts with white lines. (f) Corrected 3D model after kinematic optimization on joint position errors as described in Section VI. The offset in the arm and fingers is successfully corrected resulting in the point cloud largely overlapping with the robot model surface.

In this paper, we propose a method for *marker-less* robot arm pose estimation that is inspired by work on part-based human pose estimation [23]. We frame this problem as a classification task that labels each pixel in a depth image as representing a particular part of the robot or the background. Given these pixel labels, we use a voting scheme to re-estimate the position of each joint relative to the camera.

For the classification task we use a Random Decision

[1] Autonomous Motion Department
[2] and Perceiving Systems both at the Max-Planck-Institute for Intelligent Systems, Tübingen, Germany
[3] Computational Learning and Motor Control lab at the University of Southern California, Los Angeles, CA, USA

Forest (RDF) [3] that is trained offline on synthetically-generated data. This data contains both the 3D position of every arm joint and labels representing which part of the robot is observed in each image pixel. The data is robot-specific but can be easily generated for any other robot. In quantitative experiments, we evaluate the accuracy of the part-detection classifier. Furthermore, we show how well the approach estimates the true arm pose given these part detections on synthetic and real data. As opposed to previous work, we do not follow an iterative approach in which correspondences between the modeled and observed arm have to be estimated and their distance minimized in every iteration (e.g. Iterative Closest Point, ICP [1]). Instead, we run part detection only once per frame with a classification method that can potentially run in super-realtime [23]. The kinematically plausible arm pose is then found by gradient descent on the 3D joint positions. This results in a significantly smaller computational cost than for most ICP-based methods.

This paper is structured as follows. In the next section, we go through related work on arm pose estimation and tracking, and outline the contributions of the proposed method. Then we formally define the problem in Section III. Section IV describes how we generate the synthetic training data. Section V introduces the robot part classification framework and the features that are extracted per pixel. An explanation of the joint voting scheme and the method to re-estimate a kinematically plausible arm pose is provided in Section VI. Finally, in Section VII we evaluate the performance of the robot part classification as well as the arm pose estimation.

II. RELATED WORK

There exist quite an extensive body of work on (articulated, rigid or even deformable) object tracking and pose estimation. It is beyond the scope of this paper to review these approaches. Instead we concentrate on robot pose estimation and tracking methods. These systems usually have access to an initial estimate given by joint encoder readings but are required to be real-time capable. Furthermore, the geometry of the robot parts and their kinematic structure is well known.

Marker-based tracking is popular in robotics as it makes feature extraction more robust. Examples of these systems can be found in [26, 9, 27]. Although simple, relying on fiducial markers such as LEDs, colored spheres or Augmented Reality tags has several disadvantages. First of all, the mobility of the robot arm is constrained to have at least one marker always in view. Secondly, every trackable joint should contain at least one precisely located marker. And finally, if we consider robots that are deployed in an outdoor environment, fiducials might get covered by dust and dirt. For these reasons, we focus on developing a marker-less arm pose estimation system. Examples for systems like this are presented in our previous work [10], or in [14, 12]. These approaches employ variants of the ICP [1] algorithm. They iterate between finding a transformation that minimizes the distance between corresponding features and actually establishing these correspondences given a transformation. In our previous work [10], we use oriented edge segments of the robot arm silhouette matched with Canny edges in the camera image. Image edges have previously been used in [6, 24]. These features are efficient to compute such that real-time performance can be achieved. However, due to the use of pure 2D information, this method is sensitive to background edges or poor initialization. Krainin et al. [14] rely mostly on 3D point information but also on sparse feature and color matching. Finding the transformation that results in a minimum error computed over all this information is quite expensive. The authors estimate that a frame rate of 10Hz can be achieved with an optimized implementation. Also, Hebert et al. [12] present an ICP-based system for simultaneous arm and object tracking. Similar to [14], the authors combine information from 3D point clouds, arm and object silhouettes as well as sparse features. For efficiency, the arm model is simplified to consist of 3 cylinders. Using this, the authors achieve real-time performance. Both methods [14, 12] employ a Bayesian filter to smooth the arm and object pose estimates over time. Being tracking systems, these approaches can suffer from bad initialization or loss of tracking, while our system estimates the robot pose independently for each frame.

In comparison to these approaches, the method proposed in this paper does not rely on computing dense feature correspondences at every ICP iteration. Instead, for each incoming frame we classify each pixel only once to belong to a specific part of the robot arm. As a classifier, we use a Random Decision Forest [3] that is trained on synthetically-generated data. Each pixel is pushed through the classification trees, to end up in specific leaves which will cast a vote for joint locations. Pooling these votes over all the classified pixels will result in a set of newly estimated joint positions. Estimating the robot's true pose that matches the visually estimated joint positions only requires to compute the distances between n pairs of corresponding points where n is the number of degrees of freedom (DoF). Correspondences do not need to be estimated. In our approach, they are known and consist of pairs each containing a visually estimated joint position and the joint position from the forward kinematics.

The idea of pixel-wise part classification is inspired by the work of Shotton et al. [23] for human pose estimation from a single depth image. Due to the simplicity of the computed features and the intrinsic parallelism of pixel classification, their approach runs in super realtime on consumer hardware. In this paper, we demonstrate that the idea of pixel-wise part classification can be used for robust robot arm tracking. In comparison to robot arm tracking, human pose estimation presents the additional challenges of shape variation across people, lack of an initial pose estimate and of an accurate kinematic tree. However, a robotic arm tracker is required to perform at a much higher accuracy especially for the end effectors. Moreover, the similarity between left and right robotic arms has to be disambiguated effectively despite the full robot not being in view. We address these challenges by enforcing kinematic constraints and using the initial pose estimate. Furthermore, Shotton et al. [23] assume a given segmentation of the human from the background. We remove this assumption by introducing an additional background class. We show that after training with labeled images containing a random background, the classifier is able to distinguish between robot parts and background.

III. PROBLEM FORMULATION

The problem addressed in this paper is to compute an estimate $\hat{\theta}$ of the true joint angles $\theta = [\theta_1 \ \dots \ \theta_n]^T$. of a robot with n DoFs. Using the forward kinematics of the robot, we can compute the 3D positions $\mathbf{p}_j(\theta), \ j = 1 \dots n$ of each joint j relative to the robot base. Furthermore, we know the Jacobian Matrices $\mathbf{J}_j$ that relate changes in joint angles to changes in 3D joint positions:

$$\Delta \mathbf{p}_j(\theta) = \frac{\partial \mathbf{p}_j(\theta)}{\partial \theta} \Delta\theta = \mathbf{J}_j \Delta\theta. \tag{1}$$

A column k in a matrix $\mathbf{J}_j \in \mathbb{R}^{3\times n}$ can be interpreted as the contribution of the k^{th} joint to the change of $\mathbf{p}_j$. Specifically, if the moving joint k does not affect $\mathbf{p}_j$, its corresponding column is 0.

In this paper we propose a vision-based part detector that delivers 3D joint position estimates $\hat{\mathbf{p}}_j$ through a voting scheme. Joint encoders provide us with an initial estimate $\theta^{(0)}$ that is close to $\hat{\theta}$. The problem of finding $\hat{\theta}$ can then be posed as a least-squares problem where we want to minimize the distance to the joint position estimates

$$\min_\theta \sum_j ||\hat{\mathbf{p}}_j - \mathbf{p}_j(\theta)||^2 = \min_\theta ||\hat{\mathbf{p}} - \mathbf{p}(\theta)||^2, \tag{2}$$

with $\mathbf{p} = [\mathbf{p}_1^T \ \dots \ \mathbf{p}_n^T]^T$. Solving Eq. 2 is equivalent to solving the inverse kinematics problem and does not have a closed-form solution in general. Therefore, we solve this incrementally starting from the initial joint angles $\theta^{(0)}$. At each step, the desired increment is $\Delta\mathbf{p}^{(i)} = K(\hat{\mathbf{p}} - \mathbf{p}(\theta^{(i)}))$, with a fixed step size K. Using Eq. 1, we obtain the kinematically feasible least-squares solution

$$\Delta\theta^{(i)} = \mathbf{J}^{\dagger(i)} \Delta\mathbf{p}^{(i)} = \mathbf{J}^{\dagger(i)} K(\hat{\mathbf{p}} - \mathbf{p}^{(i)}). \tag{3}$$

$\mathbf{p}^{(i)} \in \mathbb{R}^{3n}$ are the 3D positions of all joints according to $\theta^{(i)}$, the joint angles at iteration i. $\mathbf{J}^{\dagger(i)}$ is the pseudoinverse of the joint Jacobians $\mathbf{J}^{(i)} = [\mathbf{J}_1^T \ \dots \ \mathbf{J}_n^T]^T$ as constructed from Eq. 1. These are dependent on the joint angles and therefore change at every iteration. In the context of arm tracking, Eq. 3 can also be seen as a simple proportional control law to servo the virtual arm from its initial position into the visually detected position. In this view, K is a controller gain. This has been coined by Comport et al. [6] as *virtual visual servoing*. In the next section, we will describe how we generated the synthetic training data for the pixel-wise part classifier as described in Section V. This is followed by a section on the voting scheme that delivers the visually detected 3D joint positions $\hat{\mathbf{p}}$.

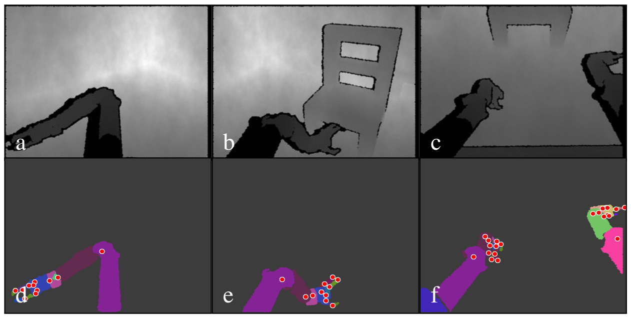


Fig. 2: Synthetically generated training data with different backgrounds. Due to the variation of the neck poses during recording, images may contain one or both arms. (a,b,c) Depth-map. (d,e,f) Ground truth robot part labels (represented with colors) and ground truth joint positions (red circles).

IV. TRAINING DATA GENERATION

The input to our robot arm pose estimation method is a depth image from an RGB-D sensor such as the Kinect or Xtion. The output is an image in which each pixel is predicted to belong to a specific robot part or the background, as well as estimates for each 3D joint position. For supervised training of such a classifier, we need depth images with ground truth part labels and ground truth joint positions.

In the case of robot arm pose estimation, we have precise CAD models of each robot part and know its kinematic structure. We can therefore avoid the tedious and inaccurate task of collecting large amounts of real data and manually labeling them. Instead, we render depth images of the robot arms in varying joint configurations using a simple ray-tracer that exploits fast intersection queries in CGAL [4]. We re-implemented the sensor model as proposed in [11] to simulate effects such as depth shadows, disparity quantization, smoothing or effects from the dot pattern that is projected into the scene. As sensor noise, we use Perlin noise [18] directly applied on the depth image. However, as pointed out in [23], the specific choice of noise model does not have significant influence on the accuracy of the classification. A sample of the generated data is shown in Fig. 2.

To allow the classifier to accurately estimate the robot pose, it needs to be trained on data that captures the variations occurring during normal operation. In our case, this concerns the different joint angles, the different viewing angles upon the scene and the varying background. In this paper, we test and demonstrate the proposed method on the ARM robot [19]. It is a dual-arm platform consisting of two 7DoF Barrett WAM arms and two 4DoF Barrett hands. When including the distal joints of the fingers and dividing the spread-angle into one joint at the root of the fingers, a total of 30 joint axis positions need to be visually estimated. Furthermore, the robot is equipped with an Xtion mounted on a pan-tilt sensor head.

We use SL [20] to simulate non-periodic joint trajectories of the two arms. Additionally at each time step, the joint angles are perturbed with Gaussian noise using a standard deviation proportional to their joint range. Labeled depth images of the resulting arm poses are generated using six different camera poses, i.e., six different sets of neck joint

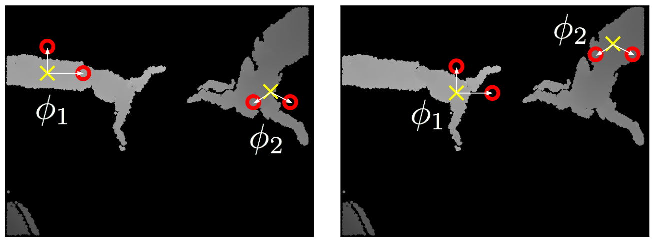


Fig. 3: Two depth image features computed at different pixels in the image. The yellow cross indicates the pixel $\mathbf{x}$ for which features are computed. The red circles indicate the relative offsets δ_u and δ_v from the pixel. (a) Features will have high values. (b) Feature value are small due to little difference in depth at the two probes.

angles. Simultaneously, we vary the poses of some furniture (two chairs, a table, closet and shelf) in the scene (a room) by randomly sampling a 2D translation and a rotation around the normal of the floor. Each sample image has a different background. as for example shown in Fig. 2.

V. PIXEL-WISE PART CLASSIFICATION

In this section, we describe the depth image features that we use to represent a pixel and briefly review random forest classifiers.

A. Depth Image Features

We employ a simplified version of the depth comparison features that were proposed in [23]. For each pixel $\mathbf{x}$ in the depth image $\mathbf{I}$, a feature ϕ_i is computed by

$$\phi_i(\mathbf{I}, \mathbf{x}) = \mathbf{I}(\mathbf{x} + \delta_u) - \mathbf{I}(\mathbf{x} + \delta_v) \tag{4}$$

where $\mathbf{I}(\mathbf{x})$ is the depth of pixel $\mathbf{x}$ in image $\mathbf{I}$. If this pixel does not contain a valid depth value, it will instead return a large positive value d_{max}. The parameters of a feature ϕ_i are the offsets δ_u and δ_v in the following referred to as depth probes. As the maximum distance of the robot arms relative to the camera is constrained by the kinematics, we dropped the depth normalization of the offsets as proposed for the original features. Fig. 3 visualizes an example in which the same features evaluate to vastly different values dependent on which pixel they are centered at. One feature alone provides only a very weak signal for discriminating parts or distinguishing foreground from background. However, a whole set $\Phi = \{\phi_0, \cdots, \phi_m\}$ of such features with depth probes randomly sampled in a window of fixed size provides a powerful signal to describe local shape variation.

B. Random Decision Forest

Random Decision Forests have proven to be powerful, fast and yet simple classifiers for multi-class problems. Apart from human pose estimation [23], they have also been used for e.g. semantic image segmentation [22] or key-point matching [15]. In this section, we will only briefly describe this method. For more detail, we refer to [3, 21].

A forest is an ensemble of T binary decision trees that consist of split and leaf nodes. Each split node is defined by one feature ϕ_i from the set Φ and an associated threshold τ. At test time, a pixel $\mathbf{x}$ from image $\mathbf{I}$ gets pushed down each tree. At each node the relevant feature centered at $\mathbf{x}$ gets evaluated according to Eq. 4 and thresholded with the associated τ. Dependent on the result, the pixel $\mathbf{x}$ continues on either the left or right branch down the tree until a leaf node is reached. A leaf node l stores a distribution $P_l(c)$ over class labels c. This distribution is modeled by a histogram computed over the class labels of the training data that ended up at this leaf node. At test time, the random forest will produce T class distributions per pixel $\mathbf{x}$. The final classification $P(c|\mathbf{I}, \mathbf{x})$ is given by averaging over these distributions.

For training the ensemble of trees, the training data set $\mathcal{Q}$ is split into T subsets $\mathcal{S} \equiv \mathcal{Q}_t$ that may be overlapping. The goal is to fit a decision tree to each $\mathcal{S}$ such that at the leaf nodes, the distribution over the classes is sufficiently *pure*. This *purity* can be defined in terms of different criteria like cross entropy, gini or misclassification rate [21]. Each split node j should divide the incoming subset $\mathcal{S}_j$ into subsets $\mathcal{S}_j^R$ and $\mathcal{S}_j^L$ so that the gain in *purity*, f, is maximized. The splitting is done according to a split function h_i from a sample set $\mathcal{H}$ of these functions. As long as the selected criteria is sufficiently improved (e.g. $f(\mathcal{S}_j, h_i) > f_{min}$) and no other stopping criteria (e.g. minimum size of the set in a leaf, S_{min}) is reached, the splitting continues. Algorithm 1 outlines the training of a tree in a recursive manner.

Algorithm 1 Training of a tree in random decision forest.

```
 1: function FITTREE($\mathcal{S}_j$, $\Phi$)
 2:     if $\#\mathcal{S}_j > S_{min}$ then
 3:         sample set $\mathcal{H}$ of split candidates $h_k = (\phi_k, \tau)$
 4:         $h_i = \arg\max_{h \in \mathcal{H}} f(\mathcal{S}_j, h)$
 5:         $\mathcal{S}_j^L = \{(\mathbf{I}, \mathbf{x}) | \phi_i(\mathbf{I}, \mathbf{x}) < \tau\}$
 6:         $\mathcal{S}_j^R = \mathcal{S}_j \backslash \mathcal{S}_j^L$
 7:         if $f(\mathcal{S}_j, h_i) > f_{min}$ then
 8:             FitTree($\mathcal{S}_j^L$, $\Phi$)
 9:             FitTree($\mathcal{S}_j^R$, $\Phi$)
10:         end if
11:     end if
12: end function
13: FitTree($\mathcal{S}$, $\Phi$)
```

VI. JOINT VOTING

In the previous section, we showed that the part classification of a pixel depends on the leaf nodes of the RDF that it ends up in. These nodes store distributions over class memberships of the training data. Apart from the ground truth part label, our training data also contains the ground truth 3D position of each joint axis. Given this information, we can compute the ground truth 3D offset from each training sample to each joint. Ideally, samples that end up at a specific leaf node should be spatially coherent. Therefore, the distribution of 3D offsets to each joint position at this leaf are expected to be clustered around a few modes. These modes represent very valuable information for estimating the offset between a pixel and a joint at test time.

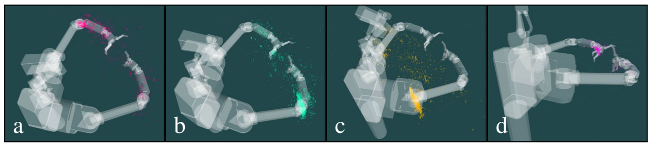


Fig. 4: Distribution of joint votes for (a) left elbow, (b) right elbow, (c) right shoulder, (d) left finger from left arm. The robot is overlaid for visualization. Although there are scattered votes, the main mass of votes concentrates around the true joint position.

A. *Learning Joint Prediction Models at Leaf Nodes*

For the problem of arm pose estimation, we use the ground truth offsets to estimate the distribution $\mathcal{R}_{lj}$ of joint offsets at each leaf node. We essentially follow the approach in [23] with a few adaptations. First of all, a training sample from a specific robot part can only cast votes for the relative position of its own and neighboring joints. Samples from the background can never vote for a joint. In this way, we avoid long-distance votes that may be subject to significantly more variation than short-distance votes. By conditioning on the kinematic structure, we also alleviate the problem of learning a per-joint distance threshold for discarding some of the votes as has been done in [23]. We cluster the resulting distributions, obtaining a set of weighted offsets $\{\mathbf{v}^i_{lj}, w^i_{lj}\}$ for each leaf l and joint j. Each $\mathbf{v}^i_{lj}$ is a 3D vector representing the mode of the offsets in cluster i and w^i_{lj} is the cardinality of this cluster. In this paper, we only use the offsets with maximum cardinality per leaf and joints: $V_{lj} = \arg\max_{(\mathbf{v}^i_{lj}, w^i_{lj})} w^i_{lj}$. The clustering is done using mean-shift [5] with a Gaussian kernel:

$$p(\mathbf{v}') \propto \sum_{\mathbf{v}\in\mathcal{R}_{lj}} \exp\left(-||\frac{\mathbf{v}'-\mathbf{v}}{b^*}||^2\right). \tag{5}$$

As bandwidth, we adopt the value of $b^* = 0.05m$ that was determined through grid search in [23]. For initialization of the clustering, we use bin-seeding with a voxel size of $2 \times b^*$. Each voxel with at least one vote serves as a seed. Mean-shift is run until convergence and clusters that are close to each other are grouped.

B. *Estimating Joint Positions*

Given the learned relative votes V_{lj} per leaf and joint, we can now apply these to estimate the joint positions at test time. For each pixel in the image, we compute the set Φ of features. After pushing them through the trained random decision forest, each data point ends up at a leaf node. For each test pixel not classified as background pixel, the associated vote V_{lj} is added to the 3D position of the test pixel. This casts a *weighted* vote for joint j. Fig. 4 shows distributions $\mathcal{P}_j$ for four example joints after all pixels that are classified as foreground have cast a vote.

The modes in this distribution constitute the 3D joint position estimates $\hat{\mathbf{p}}$. Again, we use mean-shift clustering, however this time with a kernel that takes the weights into account:

$$p(\mathbf{p}') \propto \sum_{(\mathbf{p},w)\in\mathcal{P}_j} w * \exp\left(-||\frac{\mathbf{p}'-\mathbf{p}}{b}||^2\right). \tag{6}$$

We use a shared bandwidth for all joints of $b = 0.07$ which is approximately the mean of the per-joint bandwidth values that were used in [23]. For bin-seeding we use again twice the bandwidth as voxel size. As seed threshold, we use the mean of the weights in $\mathcal{P}_j$.

VII. EXPERIMENTS

In this section, we will quantitatively evaluate the accuracy of the part classification as well as the joint voting on synthetic depth images. Furthermore, we demonstrate qualitatively the performance of the approach on real data.

A. *Training Data and Parameter Settings*

The performance of the system heavily depends on how well the training data represents the true distribution of joint angles, camera poses and scene appearance that the robot will encounter during normal operation. Two main types of data have been tested: training data with an isolated robot in the scene and training data with a randomly simulated background. The number of images generated for each of those background types were 5710 and 4818 respectively. A subset of 2000 pixels were sampled from the foreground of each image. In the data set with generated background, 1000 samples per image were additionally drawn from the background pixels.

Five trees were trained in the forest, with a minimum of 15 samples per leaf node. We selected this threshold based on a grid search on a held-out validation set (see Fig. 6). The number of features sampled in each node split was 100, and the criterion to select the best split was the Gini impurity. We used the Random Decision Forest implementation from scikit-learn [17]. Pixels with an invalid depth value were set to $d_{\max}$=3m. Due to the large size differences of the robot parts, for part classification we grouped together all links belonging to one finger and the links for the wrist pitch and yaw. For joint voting, we considered the full set of 30 joints. This grouping is detailed in Table I which also serves as a legend for the bar plot indices in this experimental section.

B. *Classification Performance*

In this section, we evaluate the performance of part classification. Specifically, we compare the two models that either assume a given segmentation or not. Furthermore, we demonstrate the performance on real-worlds examples. As a classification metric, we use the F1 score, i.e. the weighted average of precision and recall.

1) Re-Weighting the Data Set: As our data set is quite unbalanced regarding the different robot parts, we tested whether the classification accuracy improves when re-weighting the samples. Specifically, samples from smaller parts were weighted higher than samples from larger robot parts such that the sum of their weights is equal. We found no significant difference in part classification. The F1 score per part is shown in Fig. 7. Overall, the performance even slightly degrades. Fig. 5 shows qualitative examples on synthetic test data.

2) Classification Performance with and without Background Class: Segmentation of any kind of objects in a scene is a very hard problem and a research topic in itself. We felt that it is a very strong assumption to assume a segmentation of the object to be given a-priori. Therefore, we introduced an additional class in our training set: the background class of which examples are shown in Fig. 2.

We evaluated the per-part and background classification accuracy as shown in Fig. 8. In this figure, we show how the accuracy improves with an increasing training set size. It can be seen that increasing the size of the training set beyond 2700 images does not significantly improve the classification performance. The larger robot links and the background can already be classified quite reliably with small numbers of training images. The smaller links however, need more data for the classifier to be able to discriminate them. In comparison to Fig 7 that shows the classification performance if the arm is a-priori segmented, a significant drop in accuracy can be observed specifically for small robot links. This can also be observed in the qualitative examples in Fig. 9 for real data. As expected, the results that are based on a manual segmentation look better. This suggests that a second stage (as used for example in [8, 25]) in which the computed segmentation is fed into the tree could improve results with a minimal performance penalty.

C. Accuracy of Joint Position Estimates

Although the accuracy of part classification might not be as good for some smaller robot links, the distribution of joint votes at each leaf may still be spatially coherent. In this paper, we are ultimately interested in the accuracy of joint position estimates as delivered by the modes in the joint offset distribution per leaf and joint. A useful metric for the accuracy of these estimates is the mean average precision (mAP) as described in [23]. It measures how the mean area under the precision-recall curve varies with the maximum allowed distance threshold for true positive detections. It should be mentioned that we do not penalize invisible joints, i.e., joints whose associated part does not occupy more than 10 pixels in the image. For the precise definitions of this metric we refer the reader to the original publication. Figure 11a shows our results. A perfect joint position estimator would always yield mAP=1.0, i.e., per joint only the most confident vote is always inside of the true positive distance threshold. As expected, the precision of the tree requiring segmented data is better than the one of the tree that is trained without pre-segmentation. However, the decrease is not as big as for the classification. In Figure 11b this average precision is broken down per joint, for a distance threshold of 0.05 meters. Joints of smaller links show again a lower performance than the joints of larger links. Fig. 10 shows an example of the joint position estimates on real data. These estimates could now be used to find kinematically consistent robot arm poses as described in Sec. III. Fig. 1 f) shows an example re-estimation of the robot arm pose given the estimated joint positions. We additionally enforced joint limits and re-weighted the error term in Eq 2 with the confidence values of the estimates which proofed more robust against noisy estimates (see video at `http://youtu.be/xXkV6UcMCqw`). Please note that this is the first place in the framework where we make use of the information from the joint encoders. For all the other parts, i.e. classification and joint position estimation, we only used the offline-trained random decision forest and the learned voting distributions per leaf and joint.

TABLE I: Indices of Robot Links and associated Robot Parts

0	0	`L_Forearm`	1	1	`L_Left_Finger_1`
2	-	`L_Left_Finger_2`	3	-	`L_Left_Finger_3`
4	2	`L_Middle_Finger_2`	5	-	`L_Middle_Finger_3`
6	3	`L_Right_Finger_1`	7	-	`L_Right_Finger_2`
8	-	`L_Right_Finger_3`	9	4	`L_Lower_Wrist`
10	-	`L_Shoulder`	11	5	`L_Up_Arm`
12	-	`L_Up_Arm_In`	13	6	`L_Up_Wrist_Pitch`
14	-	`L_Up_Wrist_Yaw`	15	7	`R_Forearm`
16	8	`R_Left_Finger_1`	17	-	`R_Left_Finger_2`
18	-	`R_Left_Finger_3`	19	9	`R_Middle_Finger_2`
20	-	`R_Middle_Finger_3`	21	10	`R_Right_Finger_1`
22	-	`R_Right_Finger_2`	23	-	`R_Right_Finger_3`
24	11	`R_Lower_Wrist`	25	-	`R_Shoulder`
26	12	`R_Up_Arm`	27	-	`R_Up_Arm_In`
28	13	`R_Up_Wrist_Pitch`	29		`R_Up_Wrist_Yaw`

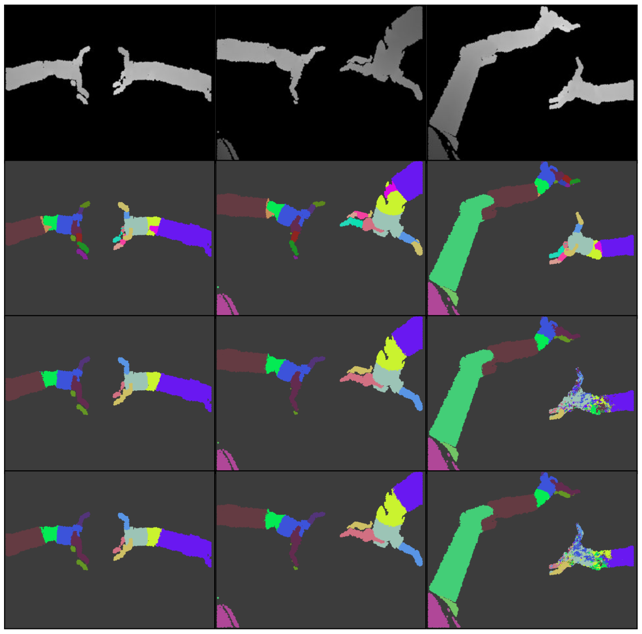


Fig. 5: Three examples (one per column) for synthetically generated labeled depth data. We can see the depth images (first row), ground truth labels (second row), and the pixel-wise part classification using unweighted (third row) and weighted (fourth row) training data. Please note that we collapsed each finger and wrist into one link only.

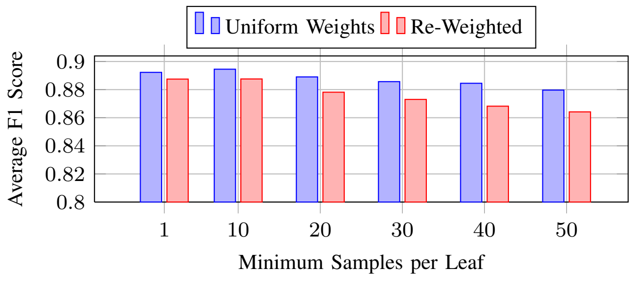


Fig. 6: Average classification accuracy over all links using different settings for minimum number of samples per leaf. Parameter search done on a held-out validation set.

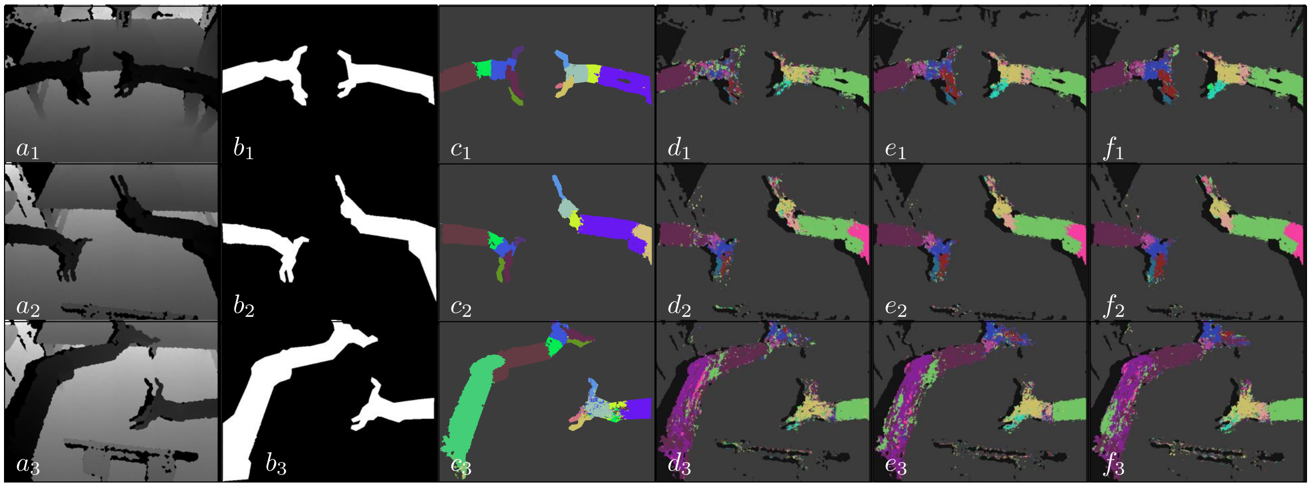


Fig. 9: Results of robot part classification on three real data examples (one per row). Column-wise we can see, (a) the real depth data, (b) a manual segmentation provided to a tree trained on pre-segmented data, (c) the result of this tree. In the following columns, we see results from a tree trained on unsegmented data with (d) 900 images, (e) 2700 images and (f) 4336 images. While the results in (c) require segmentation of the data, (d,e,f) do not. Note that the label colors are different in (c) and (d,e,f) due to the inclusion of the background label.

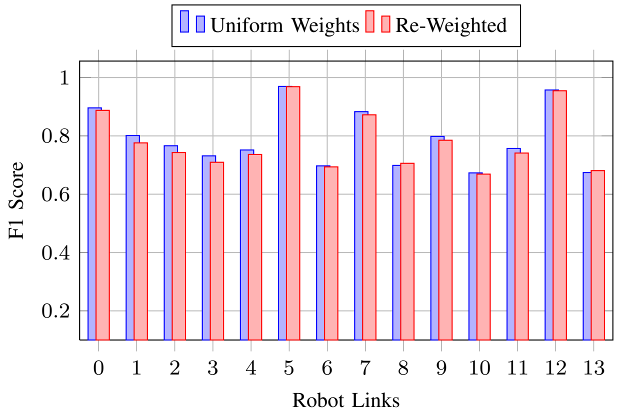


Fig. 7: Classification accuracy per part for Random Decision Forest trained with unweighted or re-weighted data.

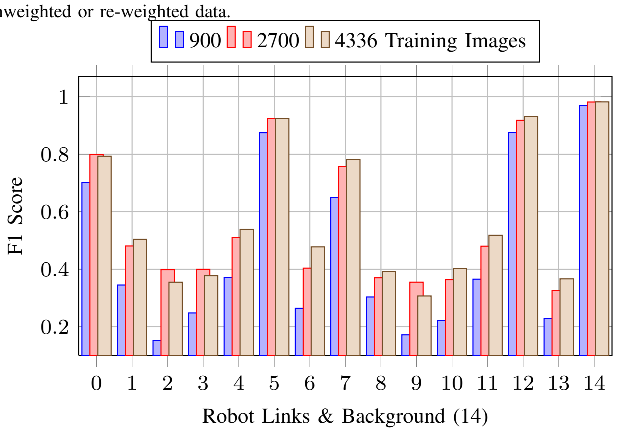


Fig. 8: Classification accuracy per part for Random Decision Forest trained on images with randomized background. The accuracy for different sizes of training data is shown.

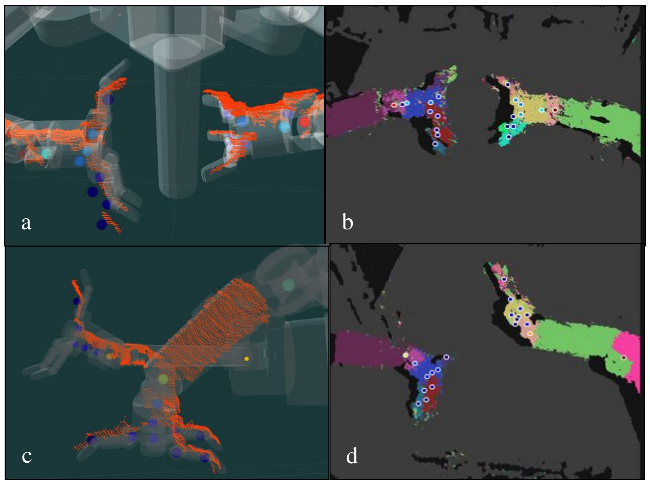
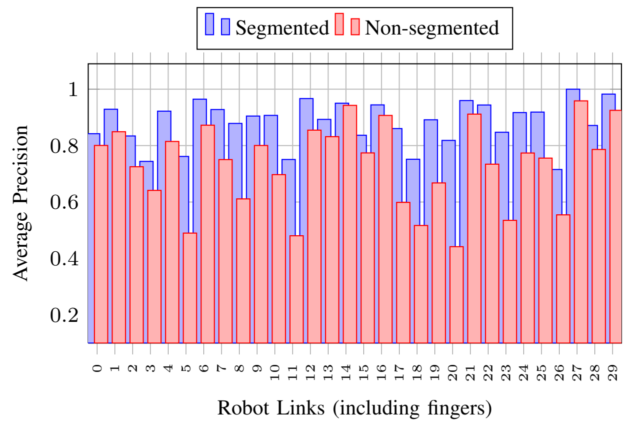


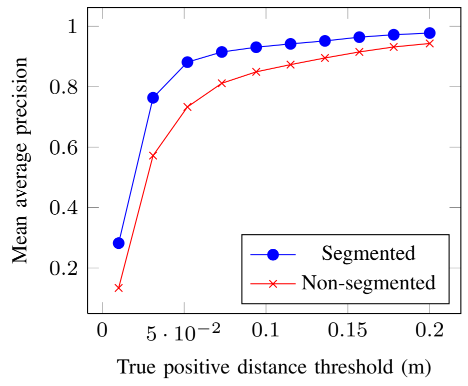
Fig. 10: Joint estimation results on real data. The first column shows the real point cloud in orange, a visualization of the robot according to the joint encoder data and the joint estimates as colored spheres. The erroneous encoder data is visualized by the lack of overalp between point cloud and robot model surface. The spheres are color-coded according to the confidence of the vote (the warmer the color the more confident the vote). The results are mostly coherent with the real data, apart from largely occluded areas like the upper left finger in (b) or the two missing fingers in (c). Fingers have much lower confidence than the wrist or elbow because the amount of pixels voting for them is much lower. By enforcing kinematic constraints, the final arm pose estimation is robust against these erroneous detections (see video at `http://youtu.be/xXkV6UcMCqw`.)

VIII. CONCLUSIONS

We have presented a method for estimating the pose of robotic manipulators from a single depth image. It is based on a pixel-wise part classification that has to be run only once per frame. Our approach is robust to sensor or feature noise (unlike edge-based systems) and potentially runs in super-realtime. Given this classification, we use a voting scheme to visually estimate 3D joint positions. These can then be used to adjust the robot pose as estimated from the joint encoder readings so that it is kinematically plausible and coherent with the recorded visual evidence. Pose estimation remains independent per frame, which makes this approach also useful as an initialization or resetting method for tracking systems like [14]. The visually estimated joint positions can be also a very valuable source for multi-feature trackers like [12]. Our current implementation is a proof-of-concept using an out-of-the-box random decision forest implementation and trivially-implemented feature extraction, resulting in non-realtime performance. However, the super-realtime visual estimation of the human skeleton on



(a)



(b)

Fig. 11: Joint estimation performance. The top figure shows the average precision (area under precision-recall curve) for each joint, computed with a maximum allowed error of 0.05 meters. The lower one shows how the mean average precision over all the joints varies with an increasing maximum allowed distance from 0.01 to 0.2 m.

an XBOX 360 (hardware from 2005) proofs that realtime performance should be easily achievable.

As can be seen e.g. in Fig. 9, the RDF that does not require pre-segmented images is able to successfully compute foreground-background segmentations. However, on pre-segmented data, the other forest performs better on part segmentation. As future work, we want to investigate whether multiple layers of random forests can exploit this fact. Similar in spirit to [8, 25], the part segmentation output of our random forest can be fed into a second random forest that exploits the segmentation input to compute more accurate joint positions. Additionally, we plan to integrate an object tracker (e.g. [27]) to track the essential elements of a grasping action. Continuous, robust tracking of both manipulators and object would allow us to apply more precise grips, manipulate moving objects and detect typical problems such as object slippage.

References

[1] P. Besl and N. D. McKay. A Method for Registration of 3-D Shapes. *IEEE Trans on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[2] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-Driven Grasp Synthesis - A Survey. *IEEE Trans on Robotics*, 2014. Accepted. PDF at `arXiv:1309.2660 [cs.RO]`.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] CGAL. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[5] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Trans on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[6] A. Comport, E. Marchand, M. Pressigout, and F. Chaumette. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. *IEEE Trans on Visualization and Computer Graphics*, 12 (4):615–628, July 2006.

[7] P. I. Corke. Visual Control Of Robot Manipulators – A Review. In *Visual Servoing*, pages 1–31. World Scientific, 1994.

[8] M. Dantone, J. Gall, C. Leistner, and L. van Gool. Human pose estimation from still images using body parts dependent joint regressors. In *IEEE-CS Conf on Computer Vision and Pattern Recognition*, 2013.

[9] X. Gratal, J. Bohg, M. Björkman, and D. Kragic. Scene representation and object grasping using active vision. In *IROS'10 Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics*, 2010.

[10] X. Gratal, J. Romero, J. Bohg, and D. Kragic. Visual servoing on unknown objects. *Mechatronics*, 22(4):423 – 435, 2012. Visual Servoing {SI}.

[11] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree. Blensor: Blender sensor simulation toolbox. In *Advances in Visual Computing*, volume 6939 of *Lecture Notes in Computer Science*, pages 199–208. Springer Berlin Heidelberg, 2011.

[12] P. Hebert, N. Hudson, J. Ma, T. Howard, T. Fuchs, M. Bajracharya, and J. Burdick. Combined Shape, Appearance and Silhouette for Simultaneous Manipulator and Object Tracking. In *IEEE Intl Conf on Robotics and Automation*, pages 2405–2412, 2012.

[13] K. Hsiao, S. Chitta, M. Ciocarlie, and E. G. Jones. Contact-reactive grasping of objects with partial shape information. In *IEEE/RSJ Intl Conf on Intelligent Robots and Systems*, pages 1228 – 1235, Taipei, Taiwan, Oct 2010.

[14] M. Krainin, P. Henry, X. Ren, and D. Fox. Manipulator and object tracking for in-hand 3D object modeling. *The Intl Journal of Robotics Research*, 30(11):1311–1327, 2011.

[15] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *IEEE-CS Conf on Computer Vision and Pattern Recognition*, volume 2, pages 775–781, June 2005.

[16] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. Online movement adaptation based on previous sensor experiences. In *IEEE/RSJ Intl Conf on Intelligent Robots and Systems*, pages 365–371, 2011.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] K. Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002.

[19] L. Righetti, M. Kalakrishnan, P. Pastor, J. Binney, J. Kelly, R. Voorhies, G. Sukhatme, and S. Schaal. An autonomous manipulation system based on force control and optimization. *Autonomous Robots*, 36(1-2):11–30, 2014.

[20] S. Schaal. The sl simulation and real-time control software package. Technical report, USC, Los Angeles, CA, USA, 2009.

[21] J. Shotton and A. Criminisi. *Decision Forests for Computer Vision and Medical Image Analysis*. Advances in Computer Vision and Pattern Recognition. Springer London, 2013.

[22] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *IEEE-CS Conf on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[23] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient human pose estimation from single depth images. *IEEE Trans on Pattern Analysis and Machine Intelligence*, 99:1, 2012.

[24] J. Sorribes, M. Prats, and A. Morales. Visual tracking of a jaw gripper based on articulated 3d models for grasping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2302–2307, 2010.

[25] Z. W. Tu. Auto-context and its application to high-level vision tasks. In *IEEE-CS Conf on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[26] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour, and R. Dillmann. Visual servoing for humanoid grasping and manipulation tasks. In *IEEE-RAS Intl Conf on Humanoid Robots*, 2008.

[27] M. Wüthrich, P. Pastor, M. Kalakrishnan, J. Bohg, and S. Schaal. Probabilistic Object Tracking using a Depth Camera. *IEEE/RSJ Intl Conf on Intelligent Robots and Systems*, 2013.