Manipulator and Object Tracking for In-Hand 3D Object Modeling

Michael Krainin[*] Peter Henry[*] Xiaofeng Ren[†] Dieter Fox[*†]

Abstract

Recognizing and manipulating objects is an important task for mobile robots performing useful services in everyday environments. While existing techniques for object recognition related to manipulation provide very good results even for noisy and incomplete data, they are typically trained using data generated in an off-line process. As a result, they do not enable a robot to acquire new object models as it operates in an environment. In this paper, we develop an approach to building 3D models of unknown objects based on a depth camera observing the robot's hand while moving an object. The approach integrates both shape and appearance information into an articulated ICP approach to track the robot's manipulator and the object. Objects are modeled by sets of surfels, which are small patches providing occlusion and appearance information. Experiments show that our approach provides very good 3D models even when the object is highly symmetric and lacks visual features and the manipulator motion is noisy. Autonomous object modeling represents a step toward improved semantic understanding, which will eventually enable robots to reason about their environments in terms of objects and their relations rather than through raw sensor data.

1 INTRODUCTION

The ability to recognize and manipulate objects is important for mobile robots performing useful services in everyday environments. Over the last years, various research groups have made substantial progress in recognition and manipulation of everyday objects (Saxena et al., 2008; Collet Romea et al., 2009; Berenson and Srinivasa, 2008; Ciocarlie et al., 2007; Lai and Fox, 2009; Rasolzadeh et al., 2009; Glover et al., 2009). While the developed techniques are often able to deal with noisy data and incomplete models, they still have limitations with respect to their usability in long-term robot deployments in realistic environments. One crucial limitation is due to the fact that there is no provision for enabling a robot to autonomously acquire new object models as it

[*]M. Krainin, P. Henry, and D. Fox are with the Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA. `mkrainin,peter,fox@cs.washington.edu`

[†]X. Ren and D. Fox are with the Intel Labs Seattle, Seattle, WA 98105, USA. `xren,dieter.fox@intel.com`

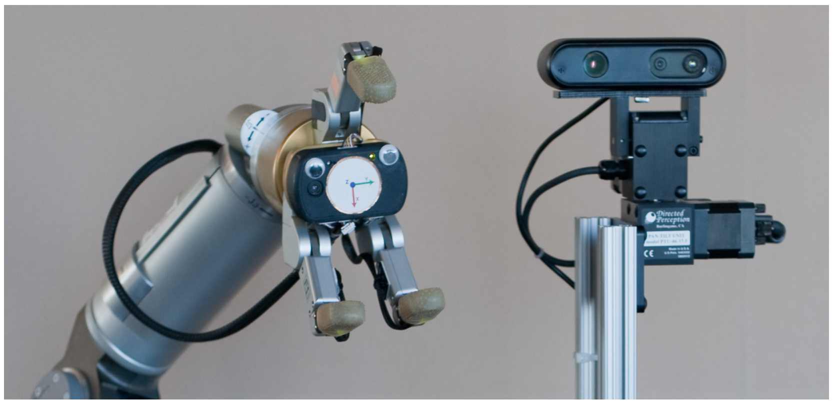


Figure 1: Experimental setup. We use a WAM arm with BarrettHand on a Segway base. Mounted next to the arm on a pan-tilt unit is a PrimeSense depth camera.

operates in an environment. This is an important limitation, since no matter how extensive the training data, a robot might always be confronted with a novel object instance or type when operating in an unknown environment.

The goal of our work is to develop techniques that enable robots to autonomously acquire models of unknown objects, thereby increasing their semantic understanding. Ultimately, such a capability will allow robots to actively investigate their environments and learn about objects in an incremental way, adding more and more knowledge over time. In addition to shape and appearance information, object models could contain information such as the weight, type, typical location, or grasp properties of the object. Equipped with these techniques, robots can become experts in their respective environments and share information with other robots, thereby allowing for rapid progress in robotic capabilities.

In this paper, we present a first step toward this long-term goal. Specifically, we develop an approach to building a 3D surface model of an unknown object based on data collected by a depth camera observing the robot's hand moving the object. In contrast to much existing work in 3D object modeling, our approach does not require a highly accurate depth sensor or a static or unobstructed view of the object, nor does it require an extremely precise manipulator. This point is essential because our manipulator can experience errors of multiple centimeters caused by cable stretch (see Fig. 3). It is also an important feature if such techniques are to be used in robots priced for consumer use.

Recently, sensors combining RGB images with depth measurements (RGB-D sensors) have come to prominence due to their gaming applications and in particular due to the release of the Xbox 360 Kinect (Microsoft, 2010). Such sensors are now both very affordable (around $150) and readily available, making them ideal for personal robotics applications. In this paper, we equip our robot with an RGB-D sensor developed by PrimeSense (a reference design equivalent to the Kinect) to allow it to sense its environment and model objects that it encounters.

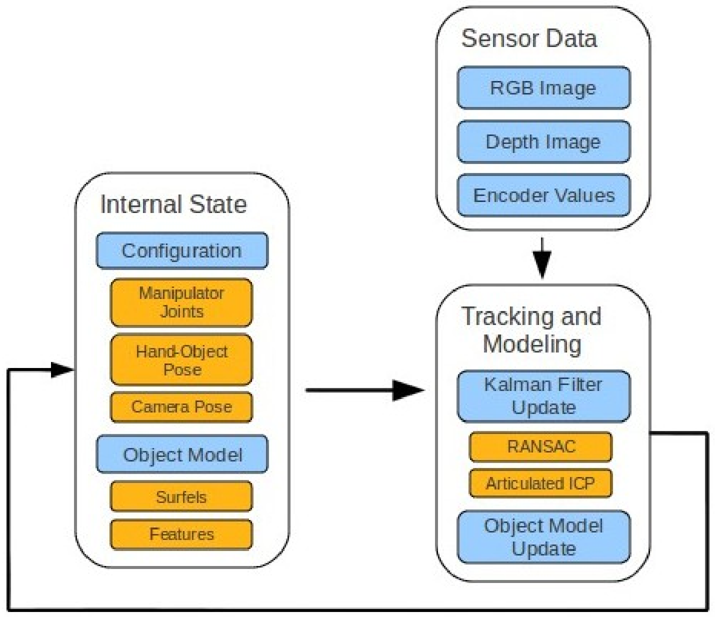


Figure 2: High level overview of our approach. As sensor data come in, we incrementally update the configuration and object model estimates.

We develop a Kalman filter that uses depth and visual information to track the configuration of the robot's manipulator along with the object in the robot's hand. By doing so, our approach can compensate for errors in manipulator and object state estimates arising from factors such as noise in the manipulator's joint sensors and poor kinematic modeling. Over time, an increasingly complete 3D model of the object is generated by extracting points from each RGB-D frame and aligning them according to the tracked hand and object position. The approach integrates the frames into a consistent surface model using surfels, small discs which represent local surface patches. Experiments show that our approach can generate good models even for objects that are highly symmetric, such as coffee cups, and objects lacking visual texture.

Fig. 2 presents a simplified outline of our approach. On each frame, new sensor data are fed into a Kalman filter, which produces pose estimates of the manipulator, object, and sensor. The Kalman filter tracking is based on RANSAC feature matching and an Iterative Closest Point variant. These estimates enable segmentation of the object from the RGB-D frame and its alignment with the existing object model. The model can then be updated based on the new sensor data.

Our work provides the following contributions:

- We propose a framework for simultaneously tracking a robotic manipulator and a grasped object while constructing a 3D model of the object. We also show how this system can be integrated with other components to perform autonomous object modeling: An unknown object can be picked up, modeled, placed back down, and regrasped so as to fill in holes caused by manipulator occlusion.

- We present a novel Iterative Closest Point (ICP) variant useful for performing tracking in RGB-D data sequences. We build upon a standard point-to-plane error metric, demonstrating extensions for sparse feature matching, dense color matching, and priors provided by a Kalman filter. Additionally, we show how to use the surfel representation to provide occlusion information to ICP.

- We propose an algorithm for merging multiple surfaces consisting of local surface patches after loop closure. Our algorithm attempts to find a consensus between the surfaces to avoid redundancy among patches.

This paper is organized as follows. In the next section, we provide an overview of related work. We then describe our Kalman filter approach to tracking a robot's manipulator and an object grasped by the hand in Section 3. We also introduce a novel version of articulated ICP suitable to our tracking task. In Section 4, we go into detail on the modeling process. Section 5 presents experimental results demonstrating the quality of both the tracking and modeling components of our algorithm. Finally, we conclude in Section 6.

2 RELATED WORK

The existing work in tracking and modeling addresses subsets of the problem we are trying to solve; however, no paper addresses them all. We make use of depth, visual, and encoder information to provide a tracking and modeling solution for enabling active object exploration for personal robotics. Below, we discuss a number of areas of research related to our own work.

2.1 Robotics for Object Modeling

Broadly speaking, object modeling techniques in robotics can be divided into two categories: ones where a sensor is moved around a stationary object and ones where the object is picked up and moved in front of a sensor. The first category avoids the difficult problem of robotic grasping and can be applied even to objects too large or delicate to be picked up. The second category, into which our technique falls, has the advantages of being able to move the object to see previously occluded sides and also lends itself to extracting further properties such as weight and stiffness.

The first category of papers is closely related to the problem of 3D mapping as it involves motion of a depth sensor in a stationary scene. Triebel et al. (Triebel et al., 2004) mount a SICK laser range finder on a four DOF manipulator for 3D volumetric modeling and exploration. They use the manipulator encoder values for sensor pose estimation. Other approaches for environment and object modeling with a depth sensor include Henry et al.'s RGB-D mapping (Henry et al., 2010) and Strobl et al.'s Self-Referenced DLR 3D-Modeller (Strobl et al., 2009). Both use visual feature tracking as the primary means of camera pose estimation. The former uses ICP to improve pose estimates, while the latter uses an IMU to provide better image flow predictions.

In the second category, Kraft et al. (Kraft et al., 2008) model contours of objects using a robotic manipulator and a stereo camera. The representations they learn, however, are not complete surface models but rather sparse sets of oriented 3D points along contours. Another important difference to our approach is that the authors assume precise camera to robot calibration and precisely known robot state at all times. We believe these assumptions to be too restrictive for the technique to be generally applicable.

Ude et al. (Ude et al., 2008) use robotic manipulation to generate training examples for object recognition. Their approach involves generating motion sequences to achieve varied views of an object, segmenting the object from images, and extracting training examples for a vision-based classifier. Unlike Kraft's work, their paper assumes neither known camera calibration nor precisely known joint angles. However, the paper does not deal with constructing 3D models and therefore does not require precise object pose.

Similarly, Li and Kleeman (Li and Kleeman, 2009) use a robotic manipulator to achieve varied views of an object for visual recognition. They store SIFT features for frames at discrete rotation angles and perform detection by matching the features of an input image against each viewpoint of each modeled object. The authors mention that such models could be useful for object pose estimation. We assert that this requires estimating the motion of the object between viewpoints using techniques such as those we propose in this paper.

2.2 In Hand Object Modeling

An area of research having a lot of recent interest is object modeling where the object is held and manipulated by a human hand. This problem is more difficult than the one we address because the there are no longer encoders to provide an (approximate) hand pose. Additionally, the appearance of human hands vary from person to person and the human hand is capable of much more intricate motions than a robotic hand. For these reasons, the hand is typically ignored. Typically, these algorithms only rely on the use of visual data or depth data but not both, and to our knowledge none explicitly try to track the hand as a means of improving alignment.

In the case of ProFORMA (Pan et al., 2009), the goal is to acquire and track models via a webcam. While visual features alone work fine for some objects, many everyday objects lack sufficient texture for this type of tracking.

Weise et al. (Weise et al., 2009) use 3D range scans and model objects using surfels but rely solely on ICP with projection-based correspondences to provide alignment. Their alignment technique is very much similar to that of Rusinkiewicz et al. (Rusinkiewicz et al., 2002) with the exception that the alignment is performed between a partial object model and the current frame rather than between the last two frames. Because they rely on geometric matching only, these techniques tend to fail for objects exhibiting rotational symmetries as many household objects do.

2.3 Articulated Tracking

A number of techniques exist for human hand-tracking; however, many of them make use of only 2D information such as silhouettes and edge detections (Athitsos and

Sclaroff, 2003; Sudderth et al., 2004). Some require pre-computed databases and may only detect configurations within that database (Athitsos and Sclaroff, 2003) and others are far from real-time algorithms. Given that we are using 3D sensors and that we wish to track the manipulator in real time through a continuous space of joint angles, such approaches are unsuitable.

In the area of human body tracking, the work of Ganapathi et al. (Ganapathi et al., 2010) is quite promising: The authors perform tracking using a time-of-flight sensor at close to real-time by taking advantage of GPUs. Their approach involves evaluating the likelihood of hypotheses by ray-tracing a model and comparing to the observed depth measurements. Additionally, parts detections can inform the search to allow recovery from failures from occlusion or fast motion. These challenges are somewhat different from those we face in our problem. Because our work focuses on robotic tracking, the motions are largely known, relatively slow, and with relatively little occlusion. Our challenge lies in precise and consistent pose estimation of the end effector and object model to facilitate object modeling. We therefore focus on incorporating many sources of information into our matching procedure rather than on fast evaluations of a purely depth-based metric.

Articulated ICP is an appealing option because of its speed, its use of 3D information, and the fact that it is readily modified to suit specific tasks, demonstrated by the wealth of existing ICP variants (see Section 2.4). It has been used in articulated pose estimation in the past (Pellegrini et al., 2008; Mündermann et al., 2007); however, to the best of our knowledge, it has not been integrated with Kalman filters, which provide the advantages of smoothing and estimating uncertainties. These uncertainties are crucial as they can be fed back into ICP to reflect the accumulated knowledge of the state (Section 3.2.3).

The work of Kashani et al. on tracking the state of heavy machinery (Kashani et al., 2010) combines particle filters with ICP. Their use of particle filters, however, is not to guide ICP's search. Rather, it provides a coarse initial registration through importance sampling over a broader set of hypotheses than ICP itself would be likely to search. It is also worth noting that the feasibility of this approach is largely due to their search space being limited to just three degrees of freedom.

2.4 ICP Variants

In this paper, we introduce a number of extensions to ICP, resulting in the error function we present in Section 3. Other works have also introduced modifications to the ICP algorithm to incorporate additional matching criteria.

A common approach is to augment each point in the two point clouds with additional attributes. The correspondence selection step then finds closest point pairs in this higher dimensional space. This approach has been applied to color (Johnson and Kang, 1997), geometric descriptors (Sharp et al., 2002), and image descriptors (Lemuz-López and Arias-Estrada, 2006). The downsides of this approach are that it requires the descriptors to be computed for every point, and the dimensionality of the nearest neighbor search increases. In comparison, our algorithm only requires SIFT descriptors at detected keypoints, and the nearest neighbor search occurs in only three dimensions.

Other approaches for incorporating additional attributes include matching only a subset of the points that have a specific attribute value (Druon et al., 2006) and constraining correspondences to be within a hard threshold with respect to attribute similarity (Godin et al., 2001). The most similar approach to ours uses projection-based correspondence selection but moves the projection along the image gradient to better match intensities (Weik, 1997). We have found that projection-based correspondence selection struggles with long, thin objects such as our robot's fingers. We therefore opt to use nearest neighbor-based correspondence selection and to augment our error function to encourage color agreement.

Feature matching has also been used to find initializations for ICP (Johnson, 1997; Lemuz-López and Arias-Estrada, 2006). Due to our tight Kalman filter integration, we already have a natural choice of initialization. We instead use our feature correspondences in the ICP error function, forcing the algorithm to consider these constraints during the final alignment.

2.5 Reconstruction

For the graphics community, obtaining accurate 3D shapes of objects is a primary research objective and has been extensively studied. Many researchers have applied range sensing of various kinds (e.g. (Curless and Levoy, 1996; Pai et al., 2001)) and can recover amazing details by combining elaborate hardware with meticulous experimental setup, such as that in the Digital Michelangelo Project (Levoy et al., 2000). In comparison, although we are recovering shape and appearance information, we do so using cheap, noisy sensors and standard robotic hardware. Our goal is to robustly and efficiently model objects and to apply such knowledge in recognition and manipulation.

In this paper we primarily perform reconstruction using a surface-element (or surfel) based approach. Surfels, originally used as a rendering primitive (Pfister et al., 2000), are oriented discs representing local surface patches on an object. They have been used with much success in reconstruction (e.g. (Habbecke and Kobbelt, 2007; Weise et al., 2009)) largely due to their conceptual simplicity and ease of implementation. In this paper, we base much of the reconstruction itself on the approach of Weise et al. We will go into more detail on surfels and surfel-based reconstruction in Section 4.

3 MANIPULATOR AND OBJECT TRACKING

Our goal is to acquire 3D models of objects grasped by a robot's manipulator. To do so, we must determine alignments between a (partial) object model and each sensor frame. Existing techniques for in-hand modeling either ignore the manipulator entirely and rely on object geometry or texture to provide alignment (Pan et al., 2009; Weise et al., 2009), or they rely on the known manipulator motion as the sole means of registration (Kraft et al., 2008; Sato et al., 1997). We argue that the first approach will fail for symmetric and/or textureless objects, while the second relies too heavily on, for example, the accuracy of the joint encoders, the kinematic modeling, and the extrinsic sensor calibration. As a demonstration of this second fact, we show in Fig. 3 that

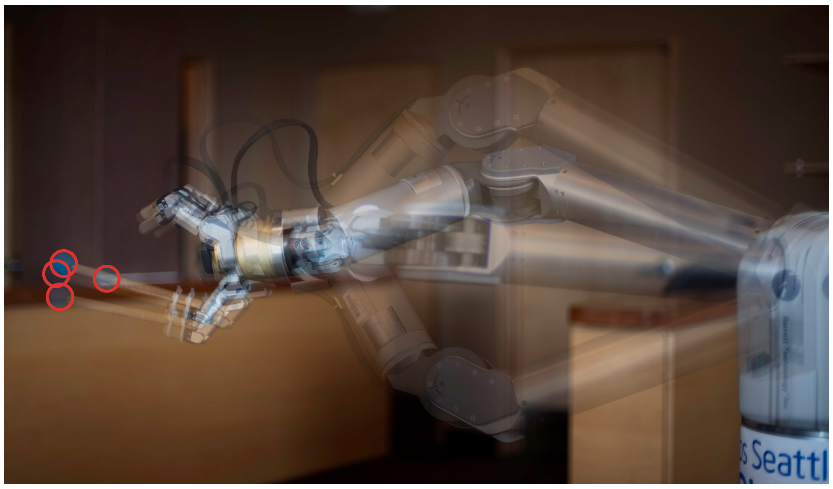


Figure 3: Pictured here is our arm in four configurations. For clarity, we have attached a ping pong ball to the end of the manipulator (highlighted with circles). In each configuration, the encoders along with forward kinematics predict the same hand pose; however, the center-to-center distance between the two farthest ball positions is approximately 8 cm.

encoder values along with forward kinematics are not always sufficiently precise for object modeling. Many of the factors contributing to the inaccuracies seen in the figure are less prominent in higher precision robots for industrial applications, but because we wish our techniques to be applicable to affordable, in-home robots, we choose not to sidestep the issue with high-precision hardware.

As an alternative to these object-tracking only and encoder only techniques, we propose to instead make use of multiple sources of information (namely encoders and RGB-D frames of *both* the manipulator and object) and to rely on each to the extent that it is reliable. We assume that the robot is equipped with a 3D depth sensor that observes the robot's manipulation space, producing 3D, colored point-clouds of the robot's manipulator and the object grasped by the hand. Fig. 4 shows an example image along with depth information of a BarrettHand holding a box. This sensor is used for both tracking and object modeling. To use such a 3D point cloud for tracking, we assume that the robot possesses a 3D model of its manipulator. Such a model can either be generated from design drawings or measured in an offline process. The 3D model allows us to generate an expected point cloud measurement for any configuration of the manipulator. In our current system, we perform a one-time ray-casting on an existing model of the WAM Arm and BarrettHand included with OpenRAVE (Diankov, 2010). In the future, we plan to investigate techniques similar to Sturm et al. (Sturm et al., 2009) to instead learn these models.

Here, we present a Kalman filter based approach having the following benefits: 1) It does not rely solely on the accuracy of the arm and is therefore applicable to a much broader class of robotic hardware. We demonstrate that our technique can function

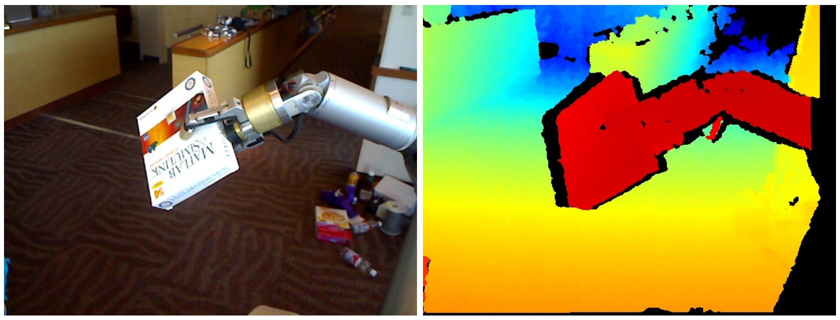


Figure 4: (left) A BarrettHand holding a box. (right) Rendering of the depth map provided by our depth camera.

even in the absence of encoder data; 2) Since the proposed technique tracks the arm in addition to the object, it can find proper alignments for geometrically- and visually-featureless object regions; 3) The algorithm reasons about the object in hand (both its motion and the occlusion it causes), which we show improves manipulator tracking; and 4) Explicitly tracking the hand leads to straight-forward hand removal from sensor data for object model update. Finally, although we focus in this paper on the benefits of the manipulator tracking technique for object modeling, the technique can be used in any situation where a more precise estimate of a robot's hand pose is desirable.

3.1 Kalman Filter Tracking

We use a Kalman filter for tracking because it helps maintain temporal consistency as well as providing estimates of uncertainty. The algorithm, shown in Table 1, takes as input the previous time step mean and covariance, surfel clouds representing the manipulator and object, and joint angles reported by the encoders of the manipulator. At a high level, the algorithm uses the measured joint angles in a motion-based update (Steps 2 - 4), uses articulated ICP to provide a measurement of the system state (Step 5), updates the object model (Step 6), and performs a measurement-based state update (Steps 7-9).

In particular, our Kalman filter state consists of three components:

- The manipulator joint angles $\hat{\theta}$.
- The transformation $\hat{T}_{calib}$, representing an adjustment to the initial robot to camera calibration, which transforms the base of the manipulator into the 3D sensor frame.
- The transformation $\hat{T}_{obj}$, representing an adjustment to the location of the object relative to the palm of the hand. It transforms the object point cloud into the reference frame of the palm.

The adjustment transformations $\hat{T}_{calib}$ and $\hat{T}_{obj}$ are initialized to identity transformations and are encoded as quaternions and translations. The initial state of the

```
1:  **Hand_object_tracker**($\mu_{k-1}, \mathbf{\Sigma}_{k-1}, \mathcal{S}_m, \mathcal{S}_{obj}, \mathcal{P}_z, \tilde{\theta}_k, \tilde{\theta}_{k-1}$):
2:  $\mathbf{u}_k = \tilde{\theta}_k - \tilde{\theta}_{k-1}$    //motion reported by encoders
3:  $\bar{\mu}_k = \mu_{k-1} + \mathbf{B}\mathbf{u}_k$  //motion update of mean
4:  $\bar{\mathbf{\Sigma}}_k = \mathbf{\Sigma}_{k-1} + \mathbf{R}_k$  //motion update of covariance
5:  $\hat{\mu}_k$ = Articulated_ICP($\mathcal{S}_m, \mathcal{S}_{obj}, \mathcal{P}_z, \bar{\mu}_k, \bar{\mathbf{\Sigma}}_k$)
6:  $\mathcal{S}'_{obj}$ = Segment_and_merge_object($\mathcal{S}_m, \mathcal{S}_{obj}, \mathcal{P}_z, \hat{\mu}_k$)
7:  $\mathbf{K}_k = \bar{\mathbf{\Sigma}}_k + (\bar{\mathbf{\Sigma}}_k + \mathbf{Q}_k)^{-1}$ //Kalman gain
8:  $\mu_k = \bar{\mu}_k + \mathbf{K}_k(\hat{\mu}_k - \bar{\mu}_k)$   //measurement update of mean
9:  $\mathbf{\Sigma}_k = (I - \mathbf{K}_k)\bar{\mathbf{\Sigma}}_k$ //measurement update of covariance
10: return $\mu_k, \mathbf{\Sigma}_k, \mathcal{S}'_{obj}$
```

Table 1: Kalman filter for joint manipulator and object tracking.

Kalman filter has associated with it a covariance matrix representing the uncertainties in the initial angles, the camera calibration, and the placement of the palm relative to the first object point-cloud.

As given in Step 2, the motion update $\mathbf{u}_k$ is based upon the difference in the reported joint angles since the previous time step. The prediction step of the Kalman filter then generates the predicted state $\bar{\mu}_k$ in Step 3. The matrix $\mathbf{B}$ simply projects the joint angle update into the higher dimensional space of the Kalman filter state. Associated with this update step is noise in the motion distributed according to the covariance matrix $\mathbf{R}_k$ (Step 4). If the camera is assumed fixed (as in our case), then $\mathbf{R}_k$ does not contribute any new uncertainty to those components of the state. The components of $\mathbf{R}_k$ representing uncertainty in the object's motion allow our algorithm to handle slight slippage of the object. In Section 3.3, we discuss how we use these components to enable regrasping the object.

If the calibration and the object's pose relative to the palm are assumed fixed (that is, if the object is grasped firmly), then $\mathbf{R}_k$ will not contribute any new uncertainty to those components of the state. Alternatively, one may include those terms in $\mathbf{R}_k$ in order to compensate for movement of the camera or the object inside the hand.

In Step 5, the function *Articulated_ICP* matches the surfel models of the manipulator and object into the observed point cloud and returns an estimate, $\hat{\mu}_k$, of the state vector that minimizes the mis-match between these clouds. Details of this algorithm are given in Section 3.2.

Segment_and_merge_object uses the output of *Articulated_ICP* to extract points from the current measurement, $\mathcal{P}_z$, that belong to the object. To do so, it uses the ICP result $\hat{\mu}_k$ to appropriately transform the manipulator surfel model $\mathcal{S}_m$ into the correct joint angles and into the sensor's reference frame. $\mathcal{S}_m$ is then used to identify points in $\mathcal{P}_z$ generated by the manipulator via simple distance checking. This hand

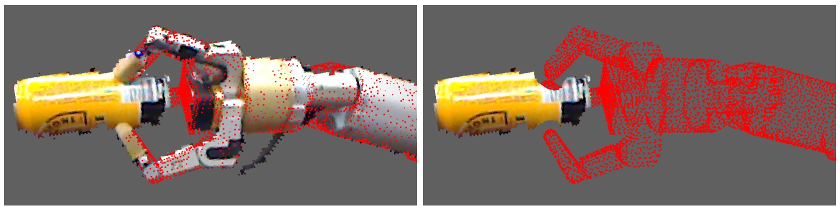


Figure 5: Because we explicitly track the robot's arm, we can remove it from the current measurement $\mathcal{P}_z$ by simply checking 3D distances to the arm model.

removal procedure is demonstrated in Fig. 5.

The points belonging to the object in the hand can then be identified due to their physical relation to the end effector. This technique has the added benefit that it does not require a static background as many vision-based algorithms do. The resulting points are then integrated into $\mathcal{S}_{obj}$ with update rules we will describe in Section 4.

Steps 7 through 9 are standard Kalman filter correction steps, where we take advantage of the fact that *Articulated_ICP* already generates an estimate of the state, $\hat{\mu}_k$, thereby allowing the simplified correction in Step 8. $\mathbf{Q}_k$ represents the uncertainty in the ICP result $\hat{\mu}_k$. While techniques do exist for estimating this matrix (e.g. by using the Jacobian matrix of the ICP error function), their estimates are based on the local neighborhood of the solution. If ICP finds only a local optimum, such estimates could drastically underestimate the degree to which the ICP solution is incorrect. We therefore set $\mathbf{Q}_k$ manually and leave online estimation as future work.

3.2 Articulated ICP

We now describe the function *Articulated_ICP* used in Step 5 of the tracking algorithm. Articulated ICP is an iterative approach to estimating joint angles by attempting to minimize an error function over two point clouds. As shown in Table 1, we use the result of the articulated ICP algorithm as a noisy state observation.

We begin with a review of the ICP algorithm for rigid objects. The input to ICP are two 3D point-clouds, a source cloud $\mathcal{P}_s = \{p_s^1, \ldots, p_s^M\}$ and a target cloud $\mathcal{P}_t = \{p_t^1, \ldots, p_t^N\}$. The goal is to find a transformation T^* (3D rotation and translation) which aligns the point-clouds as follows:

$$T^* = \operatorname*{argmin}_T \sum_{i=1}^{M} \min_{p_t^j \in \mathcal{P}_t} w_i \left| T(p_s^i) - p_t^j \right|^2 \tag{1}$$

To achieve this minimization, the ICP algorithm iterates between the inner minimization of finding correspondences as pairs of closest points given a transformation and the outer minimization of finding the transformation minimizing the sum of squared residuals given the correspondences. Since ICP only converges to a local minimum, a good initialization for T is important.

In our context, $\mathcal{P}_s$ is a combined model of the object $\mathcal{S}_{obj}$ and the manipulator $\mathcal{S}_m$, and $\mathcal{P}_t$ contains the current observation. As in (Pellegrini et al., 2008; Mündermann et al., 2007), the point clouds in our articulated ICP are related to objects that consist of multiple links connected via revolutionary joints. Specifically, each point $p_s^i \in \mathcal{P}_s$ has associated to it a link l_i in the robot's manipulator and is specified in the local coordinate system of that link. Given the state, $\mathbf{x} = \langle \theta, T_{calib}, T_{obj} \rangle$, a link l_i in the robot model has a unique transformation $T_{l_i}^S(\mathbf{x})$ that maps its points into the reference frame of the sensor. The object is treated as its own link, which has an offset T_{obj} from the palm frame. The goal of articulated ICP is to solve for the following:

$$\langle \theta, T_{calib}, T_{obj} \rangle^* = \mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{i=1}^{M} \min_{p_t^j \in \mathcal{P}_t} w_i \left| [T_{l_i}^S(\mathbf{x})](p_s^i) - p_t^j \right|^2 \tag{2}$$

We found that the use of point-to-plane type error functions (Chen and Medioni, 1992) can help to improve the performance of ICP by preventing undesired sliding of surfaces. When using point-to-plane, the ICP algorithm continues to select correspondences in the usual way (for efficiency, we use a KD-tree over the target cloud). We denote the index of the i^{th} source point's correspondence as $corr(i)$. The difference when using a point-to-plane metric is that the error function optimized by the outer minimization changes from

$$E_{pt-pt}(\mathbf{x}) = \sum_{i=1}^{M} w_i \left| [T_{l_i}^S(\mathbf{x})](p_s^i) - p_t^{corr(i)} \right|^2 \tag{3}$$

to

$$E_{pt-plane}(\mathbf{x}) = \sum_{i=1}^{M} w_i(([T_{l_i}^S(\mathbf{x})](p_s^i) - p_t^{corr(i)}) \cdot n_t^{corr(i)})^2 \tag{4}$$

To efficiently determine the normals $\{n_t^1, \ldots, n_t^N\}$ of the target cloud, we compute eigenvectors of the covariance matrices for local neighborhoods provided by the grid-structure of the data. The weight w_i for the i^{th} source point is based on the agreement between the normals of the corresponding points (the inner product of the two). Additionally, a weight is set to zero if the correspondence distance is over a threshold, if the source point's normal (provided by the surfel representation) is oriented away from the camera, or if the line segment from the camera to the source point intersects any other surfel (occlusion checking).

The point-to-plane error metric (4) is the starting point from which we extend articulated ICP. It has no closed form solution (even for the simple, non-articulated case), so we use Levenberg-Marquardt to optimize the function. We now describe extensions to the ICP error function to improve tracking by taking advantage of other available information.

3.2.1 Sparse Feature Matching

Provided we can find a set of sparse feature correspondences $\mathcal{F}$ between 3D points in the source and target clouds, we can add them to the optimization by including an extra error term:

$$E_{feature}(\mathbf{x}) = \sum_{(f_s, f_t) \in \mathcal{F}} |[T(\mathbf{x})](f_s) - f_t|^2 \tag{5}$$

Here, $T(\mathbf{x})$ refers to the transformation for the object link because, as we will explain momentarily, we only track features lying on the object. An important point to note is that (5) uses a point-to-point style error function, which provides a stronger constraint than a point-to-plane function.

These fixed, feature-based correspondences enable the use of visual information to improve matching. As an example, for objects that are geometrically featureless (e.g. planar or uniformly curved like a cup), the error function (4) is unable to disambiguate between the correct state and one in which the object model is shifted along the target surface. If the surface is textured, visual features can provide the correct within-plane shift; this is due to the use of a point-to-point error metric in (5).

Our feature-matching approach consists of maintaining a model containing locations (in the coordinate system of $\mathcal{S}_{obj}$) and descriptors of features lying on the object. In each frame, we extract a new set of visual features and determine the 3D location of the features from the depth data. We then find matches between the model features and the current frame features using RANSAC to ensure geometric consistency. Provided there are some minimum number of geometrically consistent feature matches, we use the additional error term.

To maintain the object feature model, we take an approach similar to Kawewong et al. (Kawewong et al., 2009). Features detected in each frame are matched against features from the previous frame. Only features with matches going back a specified number of frames and lying within the segmented out object are used in updating the object feature model. This avoids cluttering the model with non-stable features arising from factors such as specular highlights. Stable features with geometrically consistent matches to the feature model will result in updates to the corresponding feature in the model (running averages of position and descriptor). Stable features without matches are added into the model as new feature points. In our implementation, we use Sift-GPU (Wu, 2007) for feature detection.

3.2.2 Dense Color Matching

Sparse feature matching is a very effective technique when there are sufficiently many and sufficiently distinctive features for matching. Unfortunately, these conditions do not always hold. Not all objects have large quantities of distinctive features. Additionally, we are limited in image resolution and in the minimum distance of the object from the camera by our sensor. So for smaller objects, we typically cannot use the error term (5).

Even when distinctive visual features cannot be found, there is still useful information in the RGB image. We add the following additional error term to encourage colors in the object model to align with similarly colored pixels in the image:

$$E_{color}(\mathbf{x}) = \sum_{i=1}^{|\mathcal{S}_{obj}|} w_i \left|\left|(C \circ proj \circ T(\mathbf{x}))(p_s^i) - c_i\right|\right|_c^2 \tag{6}$$

We use the notation $(x, y) = proj(p)$ to denote the projection of a 3-D point p into the sensor's image plane and $C(x, y)$ to denote the (interpolated, or if needed, extrapolated) color value for a point in the image plane. As we will explain in further detail in Section 4, each surfel in the object model $\mathcal{S}_{obj}$ has a color associated with it. We denote the color of the i^{th} model point p_s^i as c_i. Thus, (6) penalizes differences between a surfel's color and the color at its projection.

In principal, $||.||_c$ could be any distance function over colors. In our current implementation, it is simply the magnitude of the intensity difference between the two colors. The weight w_i in (6) is 1 if the surfel has a correspondence $corr(i)$ in the point-to-plane matching (i.e. non-zero weight in (4)) and 0 otherwise. This prevents non-visible regions from being projected into the image.

Before (6) is evaluated, we first smooth the image, which has two benefits. First, it helps reduce image noise. Second, it introduces color gradients where there would otherwise be hard color boundaries. This provides the optimizer with a gradient to follow.

An important aspect to consider when performing color-based matching is that of lighting variation. One problem we came across is that specular highlights move across an object as it is being rotated. To prevent them from disturbing the color matching, we ignore any terms for which c_i or its correspondence in the target frame has an intensity over a pre-defined threshold. This heuristic is quite simple but helps prevent highlights from being drawn toward each other.

Other considerations for changes in illumination could further improve the robustness of color matching in ICP. For instance, projecting patches of points rather than one at a time would allow for the use of a metric based on normalized cross-correlation.

3.2.3 Prior State Information

Finally, we include one last error term encoding our prior knowledge about the state:

$$E_{prior}(\mathbf{x}) = (\mathbf{x} - \bar{\mu}_k)\bar{\boldsymbol{\Sigma}}_k^{-1}(\mathbf{x} - \bar{\mu}_k) \tag{7}$$

The use of this term helps influence ICP to make more coherent choices when the registration is ambiguous. For instance, if the robot rotates a solid-colored cup, a solution having the cup remain still while the hand moves will appear just as valid to the ICP error function (4) as would one in which the cup rotates along with the hand. The Kalman filter covariance $\bar{\boldsymbol{\Sigma}}_k$ encodes our uncertainties in the system's degrees of freedom and can be used to help resolve such ambiguities.

Another scenario where the use of this prior term is desirable is when the robot's hand is visible but its arm is largely out of frame. Multiple sets of joint angles give the proper hand pose and therefore the same ICP error but they are not all equally consistent with the rest of the tracking sequence.

Although there are clearly important reasons for including this error term in ICP, it should also be noted that this prior has the potential to affect the performance of the Kalman filter. $\hat{\mu}_k$ is supposed to be an independent estimate of the true state, but our prior biases it toward the existing $\bar{\mu}_k$.

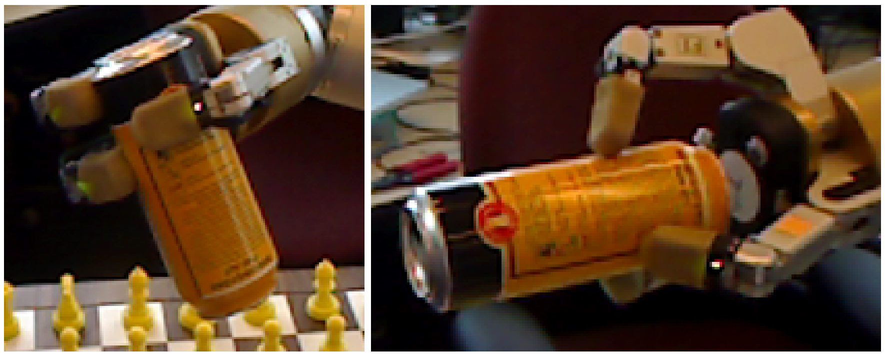


Figure 6: Two grasps of the same object. With just a single grasp, the resulting object model has holes. Any given part of the object is visible in at least one of the grasps.

With the inclusion of these additional terms, our overall ICP error function becomes

$$E(\mathbf{x}) = E_{pt-plane}(\mathbf{x}) + \alpha_1 E_{feature}(\mathbf{x}) + \alpha_2 E_{color}(\mathbf{x}) + \alpha_3 E_{prior}(\mathbf{x}) \tag{8}$$

The αs are relative weights for the components of the error function, which we set heuristically.

3.3 Handling Multiple Grasps

As described so far, the tracking procedure does not allow for regrasping of objects — an important step since the manipulator occludes parts of the object (see Fig. 6). Most notably, we assume the object moves (roughly) with the hand, which would not be true during a regrasp procedure.

To continue manipulator and object tracking during transitions between grasps, we use a switching Kalman filter with three states:

1. The robot is firmly grasping the object.
2. The robot is grasping or releasing the object.
3. The object is sitting on a table between grasps.

An advantage of performing the object modeling using a robot is that the we have knowledge of when it will grasp or release and we therefore always know which state the Kalman filter should be in.

The first state is the most common and the one we have focused on so far. The primary difference in the second state is that the object may wobble or slide in unexpected ways. We therefore modify the motion covariance $\mathbf{R}_k$ by softening the constraint on the object pose relative to the manipulator.

In the third state, the object is expected to remain fixed relative to the robot's base rather than the manipulator. In this state, the object pose components are reinterpreted as being relative to the base.

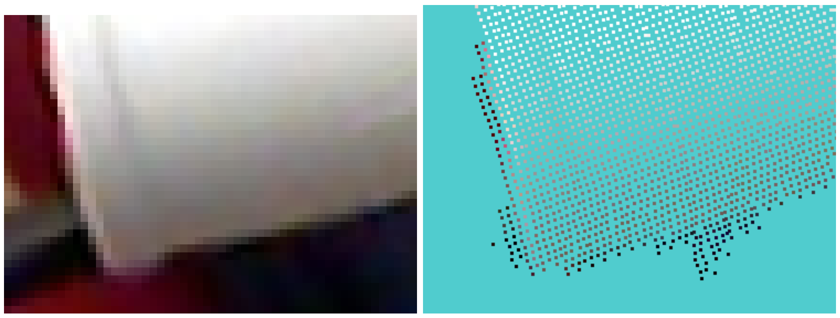


Figure 7: One of the error modes of our depth sensor. Depicted here is a point cloud of the lip of a mug against a light blue background. Along both edges shown, extra depth points appear and are colored by the scene's background. Additionally, the sensor has quantization errors and tends to fill in small holes.

The first state is the only one in which $Segment\_and\_merge\_object$ is performed. Even during the grasped state, the modeling procedure is not done while the object is being raised from or lowered to the table.

The use of a switching Kalman filter allows the robot to examine an object using one grasp, put the object down, regrasp it, and examine it again, thereby filling in holes from the first grasp. In Section 5, we demonstrate example models built from multiple grasps.

4 OBJECT MODELING

We now describe the representation underlying our object models and the key steps involved in updating object models based on new data.

4.1 Surfels

Our choice of surfels (Habbecke and Kobbelt, 2007; Weise et al., 2009) as a representation was strongly influenced by the constraints of our problem. Our depth sensor, while versatile, does suffer from certain types of noise. In particular, we must be able to compensate for quantization errors, filling in of holes, and expansion of objects by extra pixels (Fig. 7). Therefore, it is crucial that we be able to revise the models not just by adding points but also by keeping running estimates of their locations and by removing spurious points (how surfels accomplish these properties is explained in Section 4.2).

Additionally, the problem at hand involves tracking the robot's manipulator, some of which may be occluded by the object or itself. We wish to be able to reason explicitly about the visibility of any particular point in $\mathcal{S}_m$ or $\mathcal{S}_{obj}$ before assigning it a correspondence. Doing so prevents irrelevant model points from negatively impacting the alignment.

Surfels fit all of these requirements and are very easy to work with. As we explain below, the addition, update, and removal rules for surfels are quite simple and robust. While other representations such as triangle meshes could provide the occlusion information we need, the update rules can be substantially more inefficient and difficult to implement because of the need to maintain explicit connections between vertices. Surfels, on the other hand, can be updated independently of each other and if desired can be later converted to a mesh in a post-processing step.

A surfel is essentially a circular surface patch. The properties of a surfel s_i include its position p_i, its normal n_i, and its radius r_i. The radius, as described by Weise et al., is set such that as viewed from the camera position, it would fill up the area of one pixel. As the camera gets closer to the surface, surfels are automatically resized, providing an elegant means for selecting the appropriate resolution, and further, for using varying levels of detail across a single surface.

One can associate additional attributes to surfels such as "visibility confidence" v_i. The possible viewing angles of a surfel are divided into 64 bins over a hemisphere. The confidence is the number of such bins from which the surfel has been seen at least once. This provides a better measure of confidence than simply the number of frames in which a surfel has been seen because a patch seen from the same angle may consistently produce the same faulty reading.

For visualization purposes and for dense color matching (Section 3.2.2), we also keep track of the color c_i of the surfel. We use the heuristic of using the color from frame with the most perpendicular viewing angle to the surfel so far.

4.2 Model Update

After performing the segmentation described in Section 3, we use surfel update rules similar to Weise et al. (Weise et al., 2009) to modify the object model $\mathcal{S}_{obj}$. Each surfel location p_i is projected into the image plane. We then use bilinear interpolation to determine the point p_i^* and normal n_i^* at that same location in the current frame. p_i has a depth d_i and p_i^* has a depth d_i^*; the difference $d_i - d_i^*$ is used to determine the update rule that is used. In the following rules, we will say that a sensor reading p_i^* is a valid object reading if it is part of the object (as determined by the object segmentation in Section 3.1) and n_i^* does not deviate from the camera direction by more than θ_{max}.

1. $|d_i - d_i^*| \leq d_{max}$: If p_i^* is a valid object reading and n_i does not deviate from the camera direction by more than θ_{max}, then the surfel is updated. This is done by computing running averages of the surfel location and normal and updating the grid of viewing directions. Additionally, if the new measurement was taken from a closer location, then the radius of the surfel is updated accordingly. If the conditions do not hold, then we do nothing.

2. $d_i - d_i^* < -d_{max}$: In this case, the observed point is behind the surfel. If the visibility confidence v_i is below v_{high}, then the existing surfel is considered an outlier and removed. If v_i is at least v_{high}, then the reading is considered an outlier and is ignored.

3. $d_i - d_i^* > d_{max}$: Then the observed point is in front of the model surfel s_i, so we do not update the surfel.

After surfel update comes the surfel addition step. For each pixel in the object segments, a new surfel is added if there are no existing surfels with normals toward the camera either in front of or close behind the reading. This is a simple heuristic; however, it allows us to acquire models of objects which have two surfaces close together such as the inside and outside of a coffee mug. Finally, there is one more removal step. Any surfel with $v_i < v_{starve}$ that has not been seen within the last t_{starve} frames is removed. This is very effective at removing erroneous surfels without the need to return to a viewing angle capable of observing the surfel patch. More details on the parameters in this approach and reasonable values for them can be found in (Weise et al., 2009).

4.3 Loop Closure

We also base our loop closure on the techniques developed by Weise et al. (Weise et al., 2009). The approach involves maintaining a graph, whose nodes are a subset of the surfels in the object model. An edge in the graph indicates that the nodes were both visible and used for registration in the same frame. We refer the interested reader to (Weise et al., 2009) for a more detailed description of the procedure.

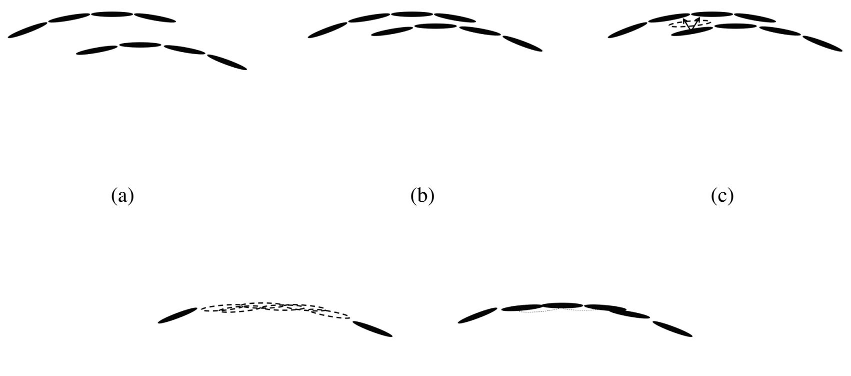
The most important operation on the graph is the computation of connected components when ignoring currently non-visible nodes. These components represent surfaces from separate passes over the visible region. To prevent all of the loop closure error from occurring in a single frame when a second connected component comes into view, only one connected component can be matched into each frame. The error is distributed over the whole structure in a separate relaxation step.

As illustrated in Fig. 8(a), the relaxation step is triggered when two connected components both individually explain some minimum percentage of the object pixels in a frame (surfels maintain connections to nearby nodes and can therefore be associated with connected components). At this point, there is some number L of connected components (usually just two), each with associated surfels. These component surfaces are surfel clouds denoted $\mathcal{S}_{C1}, \ldots, \mathcal{S}_{CL}$.

Each of the L component clouds is registered into the current frame using ICP, yielding transformations $T_{C1}^S, \ldots, T_{CL}^S$ bringing the surfels in each component into the sensor frame. The tracking algorithm described in Section 3 implicitly defines a transformation T_{obj}^S of the whole object into the sensor frame through its mean state vector. By combining these two transformations, we come to a local, adjustment transformation to each component which makes it align to the sensor data when T_{obj}^S is applied: $T_{Ci}^{adj} = (T_{obj}^S)^{-1} * T_{Ci}^S$.

We cannot simply transform each visible surfel by its appropriate T^{adj} transform because this would break the surface at the boundaries between visible and non-visible surfels. Instead, Weise et al. propose an optimization over node poses which trades off the T^{adj} constraints and relative node pose constraints.

If T_i^{init} and T_j^{init} are the pre-loop closure poses of the i^{th} and j^{th} nodes respectively, then the pose of the j^{th} node within the coordinate system of the i^{th} node is



(a) (b) (c) (d) (e)

Figure 8: Illustration of the loop closure and merging procedures. (a) Two connected components overlap enough to trigger a loop closure, (b) TORO brings the components into alignment, (c,d) Each surfel is updated to a consensus position and orientation based on nearby surfels in the other component, (e) A new surface is constructed, removing redundancy; preference is given to high confidence surfels.

$T_{i \to j}^{init} = (T_i^{init})^{-1} * T_j^{init}$. Utilizing constraints of this form in the relaxation step forces the solution to respect the relative node poses at the boundaries and to spread the error over the entire structure.

We use TORO (Grisetti et al., 2007), an open-source pose graph optimization tool, to optimize over node poses with the following constraints:

- For each node i, if it is in some connected component Cj, its pose in the object coordinate frame is constrained to be $T_{Cj}^{adj} * T_i^{init}$ with identity information matrix.
- For each node i and each of its k (we use $k = 4$) closest connected nodes j, the relative transformation between nodes is constrained to be $T_{i \to j}^{init}$ with identity information matrix.

The result of the TORO optimization is a new pose for every node in the graph. Poses of non-node surfels are subsequently determined through the interpolation process described in (Weise et al., 2009). The result will look something like Fig. 8(b).

4.4 Surface Merging

After the loop closure procedure completes, the object model will have multiple overlapping surfaces. In particular, there will be overlaps among the surfaces $\mathcal{S}_{C1}, \ldots, \mathcal{S}_{CL}$,

```
1:  **Consensus_surfel_clouds**($\mathcal{S}_1, \ldots, \mathcal{S}_L$):
2:  $\mathcal{S}'_1, \ldots, \mathcal{S}'_L = \mathcal{S}_1, \ldots, \mathcal{S}_L$
3:  for $i \in \{1, \ldots, L\}$
4:      for $j \in \{1, \ldots, |\mathcal{S}_i|\}$
5:          $depthSum = 0$
6:          $normalSum = n_j$
7:          $numSurfaces = 1$
8:          for $k \in \{1, \ldots, L\}, k \neq i$
9:              $S$ = surfels_in_range($\mathcal{S}_k$,$p_j$,$r$)
10:             if $|S| > 0$
11:                 $depthSum$ += avg_dist_along_normal($S$,$n_j$)
12:                 $normalSum$ += avg_normal($S$)
13:                 $numSurfaces$ += 1
14:         $p'_j = p_j + (depthSum/numSurfaces) * n_j$
15:         $n'_j$ = normalize($normalSum$)
16: return $\mathcal{S}'_1, \ldots, \mathcal{S}'_L$
```

Table 2: Consensus surfel algorithm. Transforms multiple surfel clouds to obtain corrected surfel positions and normals. This algorithm can be followed by a surfel removal removal step to give a single, consensus surfel cloud.

which we illustrate in Fig. 8(b). One could simply allow this overlap to persist, but it is more desirable to instead try to find a consensus among the layers to further average out measurement noise as in Fig. 8(e). A single-layered surface also avoids giving overlapping regions extra weight in (4) caused by higher point density.

Our approach is to first use the algorithm shown in Table 2 to transform each surfel into the consensus position and normal of the surfaces within a distance r. The position of a surfel is adjusted along its normal direction to match the average distance along this normal to each surface. The normal is set to the average of the normals found for each of the nearby surfaces. These averages can alternatively be weighted averages, where the weights are determined by the average visibility confidence of surfels being considered. These are omitted from Table 2 for readability. This consensus procedure is similar to the one used in mesh zippering (Turk and Levoy, 1994).

After the consensus procedure, we are faced with the problem of redundant surfels. Because it is not as straightforward to merge attributes such as viewing direction grids (which may not be aligned in their discretizations), we would like to be careful about which surfels (and therefore which attributes) we include in the updated model.

We choose to use visibility confidence as measure of which surfels to retain when there is redundancy. We construct a new surfel cloud by first sorting the surfels from all surfaces by visibility confidence. Each surfel, in order of decreasing confidence, is added if a line segment extending from $+r$ to $-r$ in its normal direction does not intersect any surfels already in the new cloud. This results in a final surfel cloud having only a single layer of surfels. A summary of the loop closure and merging procedure can be seen in Fig. 8.

5 EXPERIMENTS AND RESULTS

The robot used in our experiments is shown in Fig. 1. The basic setup consists of a WAM Arm and BarrettHand mounted on a Segway. The depth camera is located to the side and above the robot manipulator so as to provide a good view of the manipulator workspace. The specific depth camera we use, developed by PrimeSense (PrimeSense, 2010), was mainly developed for gaming and entertainment applications. It provides pixel colors and depth values at 640x480 resolution, at 30 frames per second.

We collected depth camera data and joint angle sequences of the moving system. In all but the last experiments, which use Next Best View planning, the manipulator grasps and trajectories were specified manually and the objects were grasped only once. One example of a hand-specified trajectory can be seen in Extension 1. Using multiple grasps to generate complete object models is discussed briefly in Section 5.3.

Our current implementation of the algorithm described in Section 3.1 runs at 2 to 3 frames per second. We are confident that the update rate of the system can be increased to 10 frames per second using a more optimized implementation and taking advantage of GPU hardware. To simulate such a higher update rate, we played back the datasets at approximately one fifth of the real time. We have found that by skipping frames, we are able to operate in real time, but the resulting models are not as detailed.

The tracking requires a few parameters, which we set heuristically. First are the α weights in (8). We set α_1 to 5 to give features correspondences higher weight than regular correspondences. This is desirable because we have more reason to believe the feature correspondences and because features are somewhat sparse, so there aren't as many terms in the summation. α_2 is set to .0001, which we select by giving equal weight to a 1 mm spatial error as a 10% color error, which could naturally arise from lighting variation. Finally, we set α_3 to .005, which corresponds roughly (given the typical number of correspondences we have) to equating one standard deviation from the prior to a few millimeters of misalignment for each correspondence.

For the Kalman filter, we use diagonal covariance matrices and set the variances empirically. We initialize the uncertainties in joint angles to $5\,^\circ$, camera translation and rotation to .03 m and .01 respectively (rotation uncertainty using L_2 distance of normalized quaternions), and object translation and rotation to .01 m and .01. The motion uncertainties are $2\,^\circ$ for the joints angles and .005 m and .01 for the object translation and rotation (the camera is assumed fixed). Finally, the measurement uncertainties are $4\,^\circ$ for the joint angles, .02 m and .01 for the camera translation and rotation, and .01 m and .02 for the object translation and rotation. The measurement uncertainties are purposely large to reflect that we never truly know if ICP has found the correct solution

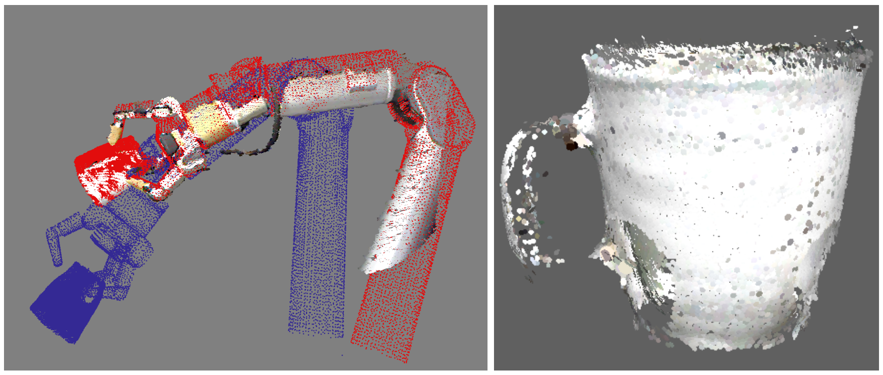


Figure 9: (left) Drift resulting from $2.0\,°/\sqrt{s}$ noise. Actual sensor data in true color, tracking result in red, and noisy joint angles in blue. (right) Surfel model produced at this level of noise.

and so the Kalman filter does not grow overconfident.

5.1 Joint Manipulator and Object Tracking

In this experiment, we evaluate the ability of our technique to track the position of the robot hand. Specifically, we investigate if our system would enable accurate tracking of a low cost manipulator equipped with position feedback far less accurate than that of the WAM arm. To do so, we use the WAM controller's reported angles as ground truth. Though these angles are far from perfect, they provide a common comparison point for the different noise settings. To simulate an arm with greater inaccuracies, we included normally distributed additive noise of varying magnitude.

To provide an intuitive feel for the units involved, we show in Fig. 9 an example of the deviation between reported and observed manipulator after 20 seconds of motion at a $2.0\,°/\sqrt{s}$ noise level. In Fig. 10, we demonstrate that our tracking algorithm can handle large amounts of noise in the reported angles without losing accuracy in tracking the end effector. Red dots in Fig. 10 show errors for the uncorrected, noisy pose estimates. Green dots, along with 95% confidence intervals, show tracking results when ignoring the object in the robot's hand. Blue dots are results for our joint tracking approach, when modeling the object and tracking it along with the manipulator. Each dot represents the end effector positioning error at the end of the tracking sequence, averaged over multiple runs and multiple arm motions. Although the noise associated with the encoder readings increases along the x-axis, we do not change the motion model covariance $\mathbf{R}_k$; however, adjusting $\mathbf{R}_k$ to better reflect the true uncertainties involved would only improve the results we present here.

As can be seen, modeling the object further increases the robustness to noise. With arm only tracking, large variations in end effector accuracy (indicative of frequent failures) begins around $3.0\,°/\sqrt{s}$, while comparable failure levels do not occur until $3.4\,°/\sqrt{s}$ for the joint tracking. This is because we can both explicitly reason about the modeled object occluding the fingers and use the object as an additional surface to

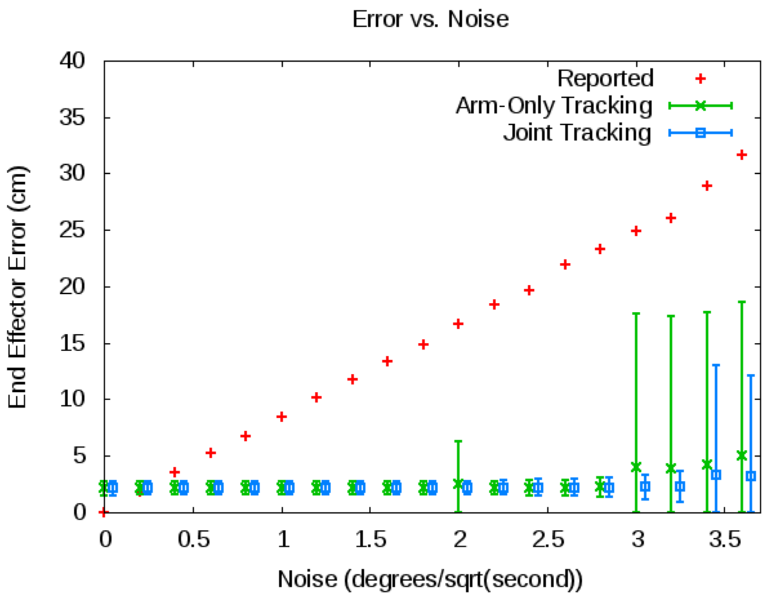


Figure 10: Distance of the end effector from the ground truth as a function of the per joint angle drift rate. Notice that the tracking errors begin earlier when only the arm and not the object is tracked. Horizontal offset of 'Joint Tracking' is for display purposes only.

match. An example model generated for the coffee mug under high noise conditions is shown in Fig. 9. When comparing this model to one built without additional noise (see Fig. 12), it becomes apparent that our approach successfully compensates for motion noise.

Additionally, we ran experiments on the same data sets ignoring the encoders entirely and assuming a constant velocity motion model for Step 2 of Table 1 (i.e. $\mathbf{u}_k = \mu_{k-1} - \mu_{k-2}$). This is not the same as ignoring the arm; the arm is still tracked and still provides benefits such as disambiguating the rotation of symmetric objects as described in Section 3. In these experiments, we found end effector errors of 2.15 cm, which is almost identical to the errors when using the true encoder values. This both attests to the accuracy of our ICP-based alignment procedure and suggests that our approach may be applicable to domains where encoder information is not available.

Besides factors such as cable stretch, which simply cause encoder values to disagree with the true joint angles, there may be other sources of noise. One such example is inaccuracies in the robot model. To test our algorithms resilience to model inaccuracies, we altered our robot model by shortening the upper arm by 1.0 cm. We then re-ran the experiments from Fig. 10 to determine how combinations of both modeling and encoder errors affect our tracking algorithm. These results are shown in Fig. 11. The end effector error increases to about 2.5 cm with the shortening of the arm, but our

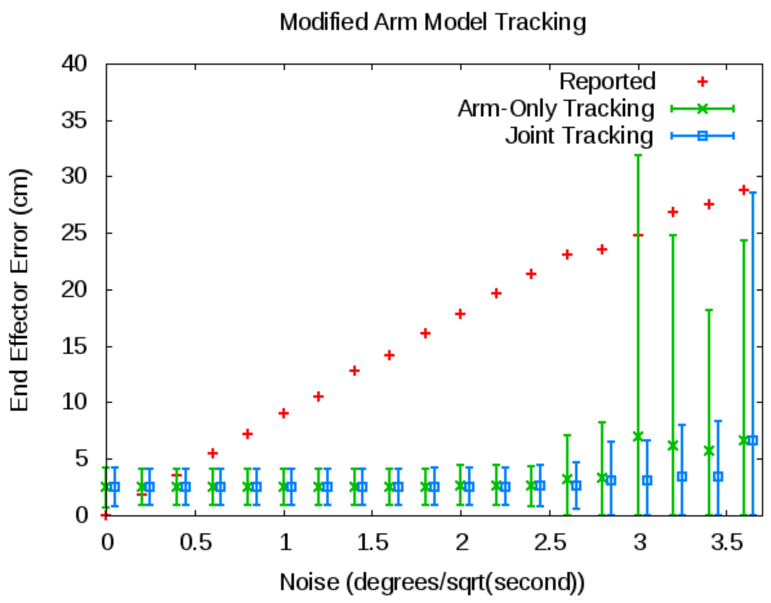


Figure 11: Tracking accuracy when using an arm model with a shortened upper arm. Our algorithm is able to track the end effector even in the presence of both robot modeling errors and discrepancies between encoder values and true joint angles.

algorithm is able to reliably track the hand up to at least $2.5\,^{\circ}/\sqrt{s}$. Notice that again, the joint tracking remains resilient to higher levels of encoder noise than the arm-only tracking.

5.2 Object Modeling

In this set of experiments, we investigate how our algorithm performs in terms of the quality of the resulting objects.

First, we examine the ability of our algorithm to model rotationally symmetric objects and objects lacking distinctive geometric regions. Many existing object modeling algorithms such as (Weise et al., 2009) rely on being able to geometrically match an object model into the new frame. In this experiment, object segmentations from our joint tracking algorithm were fed to ICP to be aligned without any information about the hand motion. The resulting clouds are compared with the output of our system in Fig. 12. As can be seen in the figure, ICP is unable to recover the proper transformations because of the ambiguity in surface matching. It should be noted that for the mug case in particular, systems like ProFORMA (Pan et al., 2009), which rely on tracking visual features would also be incapable of tracking or modeling the object.

We note that the presence of the color error term (6) can help improve the object models. The left image in Fig. 13 shows a pre-loop closure model of a can generated without the dense color error term, while the middle image shows the model built when

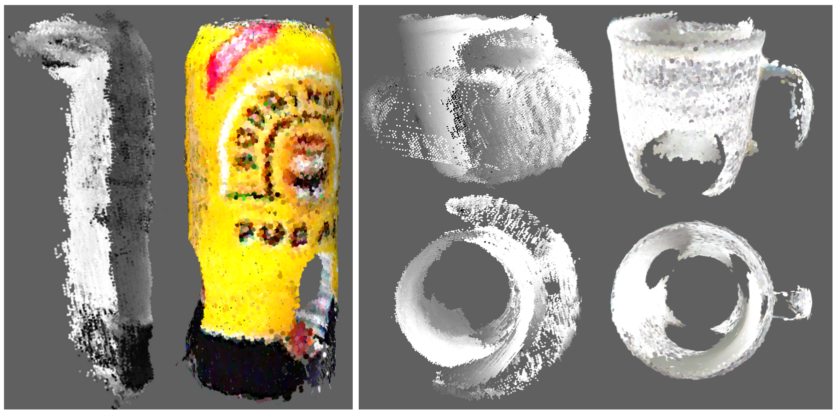


Figure 12: Shown here are a can and a mug aligned with ICP alone on the left of each image and our algorithm on the right. Due to the high level of symmetry in these objects, ICP is unable to find the correct alignments between sensor frames, resulting in largely useless object models.

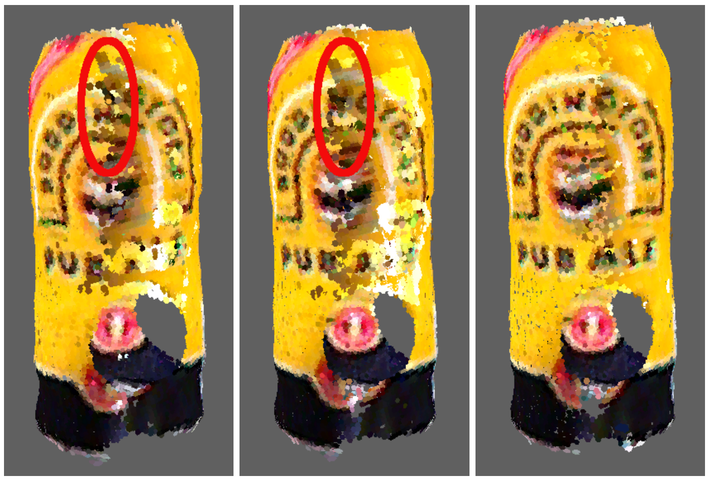


Figure 13: (left) Pre-loop closure can model without use of the dense color error term, (middle) with color error term, (right) with color error term after loop closure and surface merging.

using color. The model generated using color has noticeably less accumulated vertical misalignment.

We have also found that surfels are a very compact and elegant solution to maintaining object models. Besides the benefits of occlusion-checking and incremental update, multiple measurements can be merged into a single surfel, and the arrangement is cleaner and more visually appealing. Fig. 14 illustrates the difference between raw data

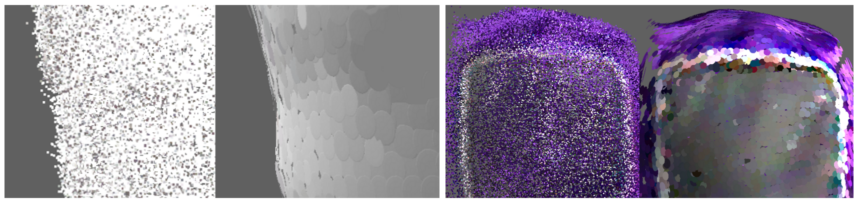


Figure 14: Comparison between aggregated point clouds and surfel models generated from the same data and the same frame alignments.

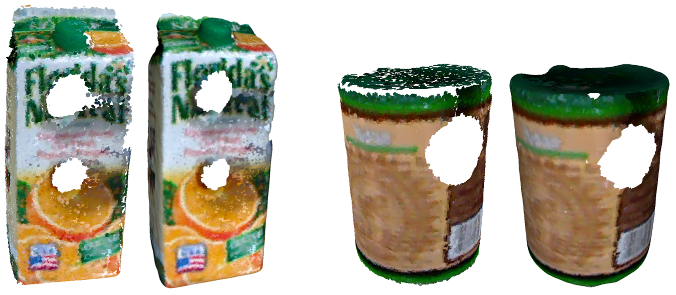


Figure 15: Surfel models and their respective meshed versions. Meshing can fill in small gaps between surfels such as the rim of the coffee can that was partially occluded by the palm. On the other hand, it struggles with thin surfaces like the top of the orange juice box and can smooth out sharp corners.

point clouds and surfel models. Shown on the right are the surfel patches belonging to two separate objects. The two panels on the left show the raw, colored point clouds from which the surfels were generated. The raw clouds contained on the order of one million points and were randomly downsampled for visualization purposes. The surfel clouds contain on the order of ten thousand surfels.

The surfel models we have obtained in our online process contain accurate information of both surface positions and normals, and can be readily converted to meshes in a post-processing step. We use the open-source Meshlab software, first applying the Poisson Reconstruction algorithm (Kazhdan et al., 2006) to obtain a surface mesh from the oriented point cloud. We then apply the Catmull-Clark subdivision to refine the mesh. Colors for vertices are assigned using the color of the nearest surfel in the original model.

Meshing is potentially useful as there are many existing algorithms which operate on meshes (e.g. simplification, refinement, hole filling). Many algorithms assume meshes as inputs, including most grasping algorithms. Meshing additionally adds visual appeal, and can fill in small gaps between surfels (and in fact larger ones, but we remove triangles with long edges to leave larger holes intact in our results). Fig. 15 shows examples of the output of the meshing procedure.

A few of the reconstructed objects are shown in Fig. 16. Rendered videos of the models are available in Extension 2. Additionally, a video demonstrating the arm track-

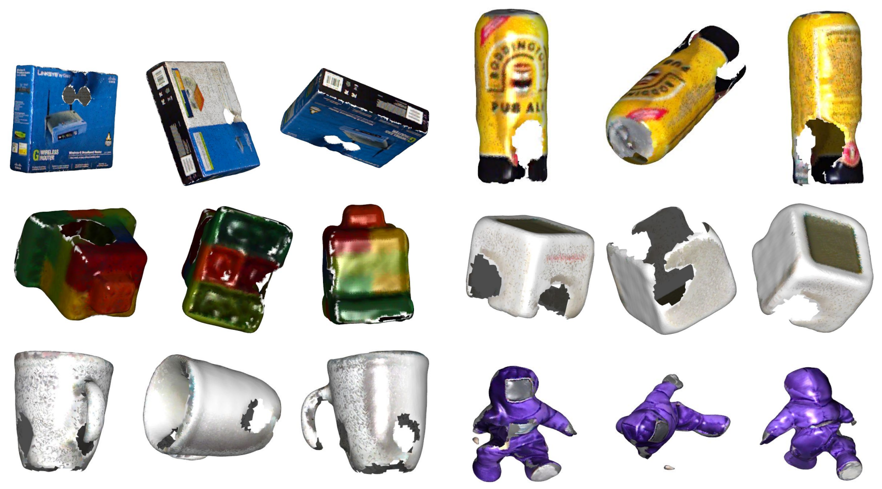


Figure 16: Triangulated surface models constructed from surfel clouds. Left column: a router box, large Lego pieces, and a mug. Right column: a can, a clock, and a stuffed doll. Holes in the models are due to occlusion by the hand or unseen regions due to the trajectory.

ing and surfel model construction is available in Extension 1.

Finally, we compare dimensions of some of the objects in Fig. 16 and Fig. 17 to measurements taken of the physical objects to better gauge the accuracy of the reconstructions. The results are shown in Table 3, and as can be seen, the reconstructed dimensions are typically within a few millimeters of the true dimensions.

Object	Dimension	Measured [cm]	Reconstructed [cm]	Error [cm]
Boddingtons Can	Diameter	6.6	6.5	0.1
Lego Block	Width	3.2	3.3	0.1
Lego Block	Length	6.4	6.7	0.3
OJ Bottle	Width	9.7	10.1	0.4
OJ Bottle	Height	19.6	19.7	0.1
Pill Bottle	Diameter	9.5	9.3	0.2
Pill Bottle	Height	18.0	17.7	0.3
White Mug	Diameter	9.2	9.4	0.2

Table 3: Comparison of reconstruction dimensions to measured dimensions of the original objects.

5.3 Toward Autonomous Object Modeling

To perform autonomous grasping and modeling, we first implemented an approach, based on the findings of Balasubramanian et al. (Balasubramanian et al., 2010), that

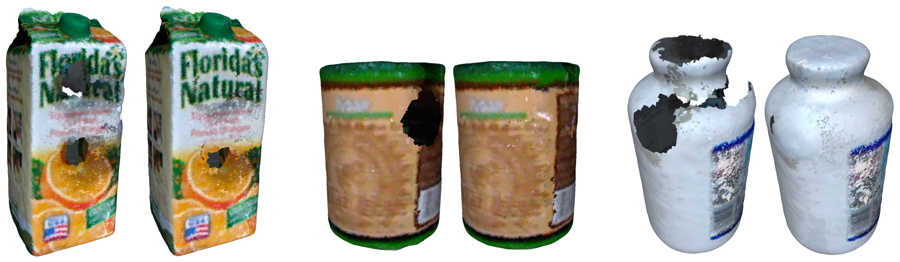


Figure 17: Side-by-side comparisons showing object models before and after regrasping. From left to right: an orange juice bottle, a coffee can, and a pill bottle. The regrasping procedure allows for holes caused by the manipulator to be filled in. In the case of the orange juice bottle, a small hole remains as there was some overlap between the grasp locations.

enables the robot to pick up an unknown object. The object grasp point and approach direction are determined by first subtracting the table plane from the depth camera data and then computing the principal component of the point cloud representing the object. The approach is then performed orthogonal to this principal component. While this technique is not intended as a general grasping approach, it worked well enough to perform our initial experiments. Alternatively, one can use local visual or geometric features as in (Saxena et al., 2008) to obtain this first grasp.

The model can be improved by allowing the robot to place the object back down and regrasp it. To generate the second grasps and to choose which regions of the object to examine, we use a Next Best View algorithm which is to appear in a follow-up paper (Krainin et al., 2011). The results of this system are shown in Fig. 17. Here it can clearly be seen that regrasping allows for filling in previously occluded regions and that the modeling is capable of resuming, even after an exchange of grasps.

Here we have presented an end-to-end robotic system, capable of picking up an unknown object, tracking and modeling the object in the presence of noisy joint sensors, regrasping the object, and resuming modeling to fill in holes. Additionally, the regrasp planning (Krainin et al., 2011) uses the partial object model, demonstrating grasp planning as one application that can greatly benefit from the ability to generate such models.

5.4 Failure Cases

Although our technique makes use of many sources of information to guide its tracking, registration failures can still occasionally occur if many things go wrong at once. One such scenario is shown in Fig. 18 (top). In these images, the front face of a box is being tracked after the box has been regrasped. The combination of an uncertain grasp location, a completely planar surface, poor lighting conditions, and an oblique viewing angle result in the misalignment.

Another place failures can occur is in the loop closure procedure, which relies on ICP to register each individual connected component into the current frame. For some

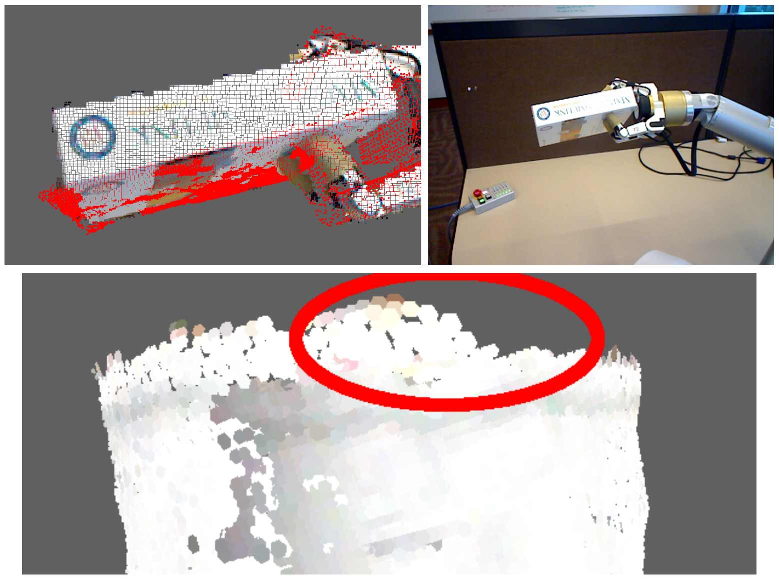


Figure 18: Failure Cases: (top-left) Poor alignment after a regrasp due to factors such as high uncertainty in the object pose, non-distinctive geometry, and poor lighting. (top-right) RGB image from the same frame. (bottom) Poorly estimated loop closure transformation on the lip of a mug (inconsistency circled) caused by fairly uniform color and geometry.

objects such as the white mug in Fig. 18 (bottom), ICP may not always succeed due to the fairly uniform color and geometry.

Finally, failures may occur during the regrasping procedure. While our tracking algorithm can handle slight wobble, sliding, or slipping during the exchange, it cannot currently handle large unexpected motions such as an object falling over on the table. This is due to ICP's requirement of being initialized close to the true state. Recovery from an object falling over would likely require a more global search.

6 CONCLUSIONS AND FUTURE WORKS

We developed an algorithm for tracking robotic manipulators and modeling grasped objects using RGB-D sensors. Our approach performs tracking, robot to sensor calibration, and object modeling in a single Kalman filter-based framework. Experiments show that the technique can robustly track a manipulator even when significant noise is imposed on the position feedback provided by the manipulator or when the robot model contains inaccuracies. The experiments also show that jointly tracking the hand and the object grasped by the hand further increases the robustness of the approach. The insight behind this technique is that even though an object might occlude the robot hand, the object itself can serve as guidance for the pose estimate.

We also introduced a tight integration of the tracking algorithm and an object modeling approach. Our technique uses the Kalman filter estimate to initially locate the object and to incorporate new observations into the object model. We use surfels as the key representation underlying the object and manipulator models. This way, our approach can do occlusion-based outlier rejection and adapt the resolution of the representation to the quality of the available data.

An approach alternative to ours could be to generate an object model by moving a camera around the object. However, this approach cannot provide information about object parts that are not visible based on the object's position in the environment. Furthermore, our approach of investigating an object in the robot's hand also lends itself to extracting information about the object's weight and surface properties.

Our key motivation for this work is in enabling robots to actively investigate objects in order to acquire rich models for future use. Toward this goal, several open research questions need to be addressed. We have implemented a very simple approach for initial grasp generation, but particularly in the presence of clutter, grasp planning for unknown objects is far from a solved problem. Also, we have shown our object models useful for grasp planning, but we would also like to explore techniques for detecting objects through attached features such as in the work of Collet et al. (Collet Romea et al., 2009) so that a robot can quickly detect and grasp objects it has examined before.

In the longer run, we would like to be able to use autonomous modeling as a step towards achieving a greater semantic understanding. For instance, if a robot can autonomously construct models of new objects and detect them reliably, it can begin extracting further information including possibly typical location, use cases, and relationships with other objects. The robot can also learn to better interact with the object by, for example, storing grasp locations that worked well on the particular object. Finally, allowing robots to share collected models and associated information would enable robots to learn much more rapidly and is a problem worth exploring.

Acknowledgements

We would like to thank Brian Mayton for all of his assistance, including generating and executing suitable grasps and arm trajectories for our experiments. We would also like to thank Peter Brook for allowing us to incorporate his heuristic grasp planner into our system.

This work was funded in part by an Intel grant, by ONR MURI grants number N00014-07-1- 0749 and N00014-09- 1-1052, and by the NSF under contract number IIS-0812671. Part of this work was also conducted through collaborative participation in the Robotics Consortium sponsored by the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016.

Extension	Media Type	Description
1	Video	A demonstration of the tracking and modeling process
2	Video	Meshed models generated from single grasps of objects

A Index to Multimedia Extensions

References

Athitsos, V. and Sclaroff, S. (2003). Estimating 3d hand pose from a cluttered image. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:432.

Balasubramanian, R., Xu, L., Brook, P., Smith, J., and Matsuoka, Y. (2010). Human-guided grasp measures improve grasp robustness on physical robot. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 2294 – 2301.

Berenson, D. and Srinivasa, S. (2008). Grasp synthesis in cluttered environments for dexterous hands. In *Proc. of the IEEE-RAS International Conference on Humanoid Robots (Humanoids)*.

Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image Vision Computing*, 10(3):145–155.

Ciocarlie, M., Goldfeder, C., and Allen, P. (2007). Dimensionality reduction for hand-independent dexterous robotic grasping. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Collet Romea, A., Berenson, D., Srinivasa, S., and Ferguson, D. (2009). Object recognition and full pose registration from a single image for robotic manipulation. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*.

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*.

Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute.

Druon, S., Aldon, M., and Crosnier, A. (2006). Color constrained icp for registration of large unstructured 3d color data sets. *International Conference on Information Acquisition*, pages 249–255.

Ganapathi, V., Plagemann, C., Koller, D., and Thrun, S. (2010). Real time motion capture using a single time-of-flight camera. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Glover, J., Rus, D., and Roy, N. (2009). Probabilistic models of object geometry with application to grasping. In *International Journal of Robotics Research (IJRR)*, volume 28, pages 999–1019.

Godin, G., Laurendeau, D., and Bergevin, R. (2001). A method for the registration of attributed range images. *3D Digital Imaging and Modeling, International Conference on*, page 179.

Grisetti, G., Grzonka, S., Stachniss, C., Pfaff, P., and Burgard, W. (2007). Efficient estimation of accurate maximum likelihood maps in 3d. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA.

Habbecke, M. and Kobbelt, L. (2007). A surface-growing approach to multi-view stereo reconstruction. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *Proc. of the International Symposium on Experimental Robotics (ISER)*, New Delhi, India.

Johnson, A. (1997). *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Johnson, A. and Kang, S. B. (1997). Registration and integration of textured 3-d data. In *International Conference on Recent Advances in 3-D Digital Imaging and Modeling (3DIM '97)*, pages 234 – 241.

Kashani, A., Owen, W. S., Himmelman, N., Lawrence, P. D., and Hall, R. A. (2010). Laser Scanner-based End-effector Tracking and Joint Variable Extraction for Heavy Machinery. *The International Journal of Robotics Research*.

Kawewong, A., Tangruamsub, S., and Hasegawa, O. (2009). Wide-baseline visible features for highly dynamic scene recognition. In *Proc. of the 13th International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 723–731, Berlin, Heidelberg. Springer-Verlag.

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Symposium on Geometry Processing*.

Kraft, D., Pugeault, N., Baseski, E., Popovic, M., Kragic, D., Kalkan, S., Wrgtter, F., and Krger, N. (2008). Birth of the object: Detection of objectness and extraction of object shape through object-action complexes. *I. J. Humanoid Robotics*, 5(2):247–265.

Krainin, M., Curless, B., and Fox, D. (2011). Autonomous Generation of Complete 3D Object Models Using Next Best View Manipulation Planning. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, Shanghai, China. (To Appear).

Lai, K. and Fox, D. (2009). 3D laser scan classification using web data and domain adaptation. In *Proc. of Robotics: Science and Systems (RSS)*.

Lemuz-López, R. and Arias-Estrada, M. (2006). Iterative closest sift formulation for robust feature matching. In *Proc. of the Second International Symposium on Advances in Visual Computing*, volume 4292 of *Lecture Notes in Computer Science*, pages 502–513. Springer.

Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D. (2000). The digital michelangelo project: 3d scanning of large statues. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*.

Li, W. H. and Kleeman, L. (2009). Interactive learning of visually symmetric objects. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St Louis, MO, USA.

Microsoft (2010). `http://www.xbox.com/en-US/kinect`.

Mündermann, L., Corazza, S., and Andriacchi, T. P. (2007). Accurately measuring human movement using articulated icp with soft-joint constraints and a repository of articulated models. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Pai, D., van den Doel, K., James, D., Lang, J., Lloyd, J., Richmond, J., and Yau, S. (2001). Scanning physical interaction behavior of 3d objects. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, pages 87–96.

Pan, Q., Reitmayr, G., and Drummond, T. (2009). ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. In *Proc. 20th British Machine Vision Conference (BMVC)*, London.

Pellegrini, S., Schindler, K., , and Nardi, D. (2008). A generalisation of the icp algorithm for articulated bodies. In Everingham, M. and Needham, C., editors, *British Machine Vision Conference (BMVC'08)*.

Pfister, H., Zwicker, M., van Baar, J., and Gross, M. (2000). Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 335–342, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

PrimeSense (2010). `http://www.primesense.com/`.

Rasolzadeh, B., Bjorkman, M., Huebner, K., and Kragic, D. (2009). An active vision system for detecting, fixating and manipulating objects in real world. In *International Journal of Robotics Research (IJRR)*.

Rusinkiewicz, S., Hall-Holt, O., and Levoy, M. (2002). Real-time 3D model acquisition. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 21(3):438–446.

Sato, Y., Wheeler, M., and Ikeuchi, K. (1997). Object shape and reflectance modeling from observation. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, pages 379 – 387.

Saxena, A., Driemeyer, J., and Ng, A. (2008). Robotic grasping of novel objects using vision. *International Journal of Robotics Research (IJRR)*, 27(2).

Sharp, G. C., Lee, S. W., and Wehe, D. K. (2002). Icp registration using invariant features. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(1):90–102.

Strobl, K., Mair, E., Bodenmuller, T., Kielhofer, S., Sepp, W., Suppa, M., Burschka, D., and Hirzinger, G. (2009). The self-referenced dlr 3d-modeler. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 21 –28.

Sturm, J., Plagemann, C., and Burgard, W. (2009). Body schema learning for robotic manipulators from visual self-perception. *Journal of Physiology-Paris*, 103(3-5):220 – 231. Neurorobotics.

Sudderth, E. B., Mandel, M. I., Freeman, W. T., and Willsky, A. S. (2004). Visual hand tracking using nonparametric belief propagation. *Computer Vision and Pattern Recognition Workshop*, 12:189.

Triebel, R., Frank, B., Meyer, J., and Burgard, W. (2004). First steps towards a robotic system for flexible volumetric mapping of indoor environments. In *Proc. of the 5th IFAC Symp. on Intelligent Autonomous Vehicles*.

Turk, G. and Levoy, M. (1994). Zippered polygon meshes from range images. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, pages 311–318, New York, NY, USA. ACM.

Ude, A., Omrcen, D., and Cheng, G. (2008). Making object learning and recognition an active process. *I. J. Humanoid Robotics*, 5(2):267–286.

Weik, S. (1997). Registration of 3-d partial surface models using luminance and depth information. *3D Digital Imaging and Modeling, International Conference on*, page 93.

Weise, T., Wismer, T., Leibe, B., , and Gool, L. V. (2009). In-hand scanning with online loop closure. In *IEEE International Workshop on 3-D Digital Imaging and Modeling*.

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