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 offline 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 lacking visual features and the manipulator motion is noisy.

1 INTRODUCTION

The ability to recognize and manipulate objects is an important task 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.,

* 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 used a WAM arm with BarrettHand on a Segway base. Mounted next to the arm on a pan-tilt unit is a depth camera.

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 the parameters of the object recognition algorithms are either set manually or trained using offline machine learning techniques. As a result, there is no provision for enabling a robot to autonomously acquire new object models as it 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 (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. 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 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 existing work in 3D object modeling (Pai et al., 2001; Curless and Levoy, 1996), our approach does not require a highly accurate depth sensor or a static or unobstructed view of the object. Neither does our approach require an extremely precise manipulator as in existing work on using robots to model objects (Kraft et al., 2008). This point is essential because our manipulator can experience multiple cm errors caused by cable stretch (see Fig. 2). 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 sen-
sensors) have come to prominence due to their gaming applications. Such sensors are now both very affordable and readily available, making them ideal for personal robotics applications. We equip our robot with one of these sensors to allow it to sense its environment and model objects that it encounters.

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 depth scan and aligning them according to the tracked hand and object position. The approach integrates the scans into a consistent surface model using surfels, small discs which represent local surface patches (Habbecke and Kobbelt, 2007; Weise et al., 2009). 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.

Our work provides the following contributions:

- We demonstrate a robotic system bringing together and building upon techniques such as surfel-based object modeling and articulated tracking. Our system is capable of grasping an unknown object, acquiring a model of the object, and placing it back on the table. We also show that our technique is capable of resuming modeling after the object is placed down and regrasped.

- 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. Our algorithm attempts to find a consensus between the surfaces and to avoid throwing away data when possible.

This paper is organized as follows. In the next section, we first describe our Kalman filter approach to tracking a robot’s manipulator and an object grasped by the hand. We then introduce a novel version of articulated ICP suitable to our tracking task. We then go into detail on the modeling process in Section 3. Then, in Section 4, we discuss related work, followed by experimental results in Section 5. Finally, we conclude in Section 6.

2 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 rely on the known manipulator motion as the sole means of registration
Figure 2: Pictured here is our arm in four configurations. For clarity, we have attached a ping pong ball to the end of the manipulator (highlighted in red). 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 8cm. (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 the accuracy of joint encoders, kinematic modeling, calibration, and other factors. As a demonstration of this second fact, we show in Fig. 2 that encoder values along with forward kinematics are not always sufficient for object modeling. Many of the factors contributing to the inaccuracies seen in the figure are less prominent in industrial-quality robots, 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.

We propose as an alternative to these object-tracking only and encoder only techniques 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. 3 shows an example image along with depth information of a BarrettHand holding a box. This sensor is used both for tracking and for object modeling. To use such a point cloud for tracking, we assume that the robot has a 3D model of its manipulator. Such a model can either be generated from design drawings or measured in an off line 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. 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 will demonstrate that our technique can function even in the absence of encoder data; 2) Because 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 presence of the object in the hand is beneficial to rather than detrimental to the arm tracking as it provides additional regions for geometric and visual matching, and the algorithm explicitly reasons about any occlusion caused by the object; 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, it should be noted that it is more broadly useful and can be used in any situation where a more precise estimate of a robot’s hand pose is desirable.

2.1 Kalman Filter Tracking

We use a Kalman filter to track the three components in the state vector \( \mu = (\hat{\theta}, \hat{T}_{calib}, \hat{T}_{obj}) \):

- 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 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.
1: **Hand_object_tracker**(µk−1, Σk−1, Sm, Sobj, Pz, θ̂k, θ̂k−1):

2: \( u_k = \tilde{\theta}_k - \tilde{\theta}_{k-1} \)

3: \( \bar{\mu}_k = \mu_{k-1} + Bu_k \)

4: \( \bar{\Sigma}_k = \Sigma_{k-1} + R_k \)

5: \( \hat{\mu}_k = \text{Articulated}_\text{ICP}(S_m, S_{\text{obj}}, P_z, \bar{\mu}_k, \bar{\Sigma}_k) \)

6: \( S'_{\text{obj}} = \text{Segment}_\text{and_merge}_\text{object}(S_m, S_{\text{obj}}, P_z, \hat{\mu}_k) \)

7: \( K_k = \bar{\Sigma}_k + (\bar{\Sigma}_k + Q_k)^{-1} \)

8: \( \mu_k = \bar{\mu}_k + K_k(\hat{\mu}_k - \bar{\mu}_k) \)

9: \( \Sigma_k = (I - K_k)\bar{\Sigma}_k \)

10: return \( \mu_k, \Sigma_k, S'_{\text{obj}} \)

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

The algorithm, shown in Table 1, takes as input \( \mu_{k-1} \) and \( \Sigma_{k-1} \), the previous time mean and covariance. Additionally, \( S_m \) and \( S_{\text{obj}} \) are surfel clouds representing the manipulator and the object respectively. As we will explain in more detail below, surfels describe small patches on the surface of an object, thereby providing more information than pure point clouds. Initially, \( S_{\text{obj}} \) is empty since no model of the object is available. \( \tilde{\theta}_k \) and \( \tilde{\theta}_{k-1} \) are the joint angles reported by the encoders of the manipulator.

As given in Step 2, the motion update \( u_k \) is based upon the difference in reported joint angles since the previous timestep. The prediction step of the Kalman filter then generates the predicted state \( \bar{\mu}_k \) in Step 3. The matrix \( B \) 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 \( R_k \) (Step 4). If the calibration and the object’s pose relative to the palm are assumed fixed (that is, if the object is grasped firmly), then \( R_k \) will not contribute any new uncertainty to those components of the state. Alternatively, one may include those terms in \( R_k \) in order to compensate for movement of the camera or the object inside the hand.

In Step 5, the function \( \text{Articulated}_\text{ICP} \) 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 2.2.

\( \text{Segment}_\text{and_merge}_\text{object} \) uses the output of \( \text{Articulated}_\text{ICP} \) to extract points from the current measurement, \( P_z \), that belong to the object. To do so, it uses the ICP result \( \hat{\mu}_k \) to appropriately transform the manipulator surfel model \( S_m \) into the correct joint angles and into the sensor’s reference frame. \( S_m \) is then used to identify points in
generated by the manipulator via simple distance checking. After segmenting the remaining points in \( P_z \) (which can be done by computing connected components over the grid structure of the point cloud), the points belonging to the object in hand can 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 \( S_{obj} \) with update rules we will describe in Section 3.

Steps 7 through 9 are standard Kalman filter correction steps, where we take advantage of the fact that \textit{Articulated ICP} already generates an estimate of the state, \( \hat{\mu}_k \), thereby allowing the simplified correction in Step 8. \( Q_k \) 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 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 \( Q_k \) by hand and leave online estimation as future work.

2.2 Articulated ICP

We now describe the function \textit{Articulated ICP} used in Step 5 of the tracking algorithm. We begin with a review of the ICP algorithm for rigid objects. The input to ICP are two 3D point-clouds, a source cloud \( P_s = \{ p_1^s, ..., p_M^s \} \) and a target cloud \( P_t = \{ p_1^t, ..., p_N^t \} \). The goal is to find a transformation \( T^* \) (3D rotation and translation) which aligns the point-clouds as follows:

\[
T^* = \arg\min_T \sum_{i=1}^M \min_{p_i^t \in P_t} w_i \left| T(p_i^s) - p_i^t \right|^2
\]

(1)

To achieve this minimization, the ICP algorithm iterates between the inner minimization of finding correspondences 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, \( P_s \) is a combined model of the object \( S_{obj} \) and the manipulator \( S_m \), and \( P_t \) 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 \( p_i^s \in P_s \) 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, \( x = (\theta, T_{calib}, T_{obj}) \), a link \( l_i \) in the robot model has a unique transformation \( T_{l_i}^S(x) \) 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:

\[
(\theta, T_{calib}, T_{obj})^* = x^* = \arg\min_x \sum_{i=1}^M \min_{p_i^t \in P_s} \left| T_{l_i}^S(x)(p_i^s) - p_i^t \right|^2
\]

(2)

We have found that the use of point-to-plane type error functions (Chen and Medioni, 1992) can help 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 $\text{corr}(i)$. The difference when using a point-to-plane metric is that the error function optimized by the inner minimization changes from

$$E_{pt-pt}(x) = \sum_{i=1}^{M} w_i \left[ \| T^S_{i}(x) (p^s_i) - p^t_{\text{corr}(i)} \|_2^2 \right]$$

(3)

to

$$E_{pt-plane}(x) = \sum_{i=1}^{M} w_i \left( \| [T^S_{i}(x)](p^s_i) - p^t_{\text{corr}(i)} \|_2^2 + (\vec{n}^t_{\text{corr}(i)} \cdot \vec{n}^s_{\text{corr}(i)}) \right)^2$$

(4)

To efficiently determine the normals $\{n_1^t, \ldots, n^N_t\}$ 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. 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.

### 2.2.1 Sparse Feature Matching

For objects that are geometrically featureless (e.g. planar or uniformly curved), 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. In these cases, we would like to be able to use visual information to determine the correct registration.

Visual features have often been used with good success to estimate 6D pose for textured objects (Collet Romea et al., 2009; Kuehnle et al., 2009; Pan et al., 2009; Azad et al., 2009). To incorporate this type of matching into our tracking framework, we maintain a model (described below) of the 3D locations of observed features on the object model. In each frame, we extract a new set of visual features and determine the 3D location of the feature point from the depth data. We then find matches between the model features and the current frame features using RANSAC to ensure geometric consistency. The result of RANSAC is a set $F = \{(f^s_i, f^t_i), \ldots, (f^s_i, f^t_i)\}$ of correspondences between model feature locations $f^s_i$ and current frame feature locations $f^t_i$. The correspondences are straightforward to add into ICP as fixed correspondences by including an extra error term:

$$E_{feature}(x) = \sum_{(f^s_i, f^t_i) \in F} \| T^s(x) (f^s_i) - f^t_i \|^2$$

(5)
Here, $T(x)$ refers to the transformation for the object link. Another point to note is that we only use this error term if $F$ has at least some minimum number of correspondences. By requiring a minimum number of geometrically consistent correspondences, we can be confident in the veracity of the RANSAC inliers. Additionally, the use of a point-to-point style error function provides a stronger constraint than the point-to-plane terms in (4) and is therefore able to resolve the within plane sliding that is otherwise ambiguous.

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 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, normal, and descriptor). Stable features without matches are added into the model. In our implementation, we use SiftGPU (Wu, 2007) for feature detection.

2.2.2 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 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 shown in (5).

Still, we would like to be able to use visual information to constrain the object pose even when we cannot determine precise correspondences from feature matching. To accomplish this, we consider the projections of points in the object model onto the image plane. 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 3, each surfel in the object model $S_{obj}$ has a color associated with it. We will denote the color of the $i^{th}$ model point $p^i$ as $c_i$.

We then define the dense color matching error as:

$$E_{color}(x) = \sum_{i=1}^{S_{obj}} w_i \cdot |C \circ proj \circ T(x)(p^i) - c_i|^2$$

(6)

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. The first is that it helps remove noise. Secondly, it introduces color gradients where there would
otherwise be hard color boundaries. This helps provide 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 have run across is that specular highlights will move across an object as it is rotated. To help prevent these from disturbing the color matching, we ignore any correspondence for which \( c_i \) or \( C(\text{proj}(\hat{t}^\text{corr}(i))) \) 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 help 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.

### 2.2.3 Prior State Information

Finally, we would like to be able to use our existing knowledge about the state of the system to influence ICP to make the more likely 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 (as described so far) as would one in which the cup rotates along with the hand.

The Kalman filter covariance \( \bar{\Sigma}_k \) encodes our uncertainties in the system’s degrees of freedom and can be used to help resolve such ambiguities. In the above example, we would like ICP to be able to determine from the probability distribution that there is only a small uncertainty in the relative transformation between the hand and the cup (assuming a fairly solid grasp) and therefore the more likely occurrence is that the cup has moved according to the hand’s motion.

The use of such prior knowledge can also help in the opposite way. If the object pose can be very well estimated due to, for instance, strong visual features, it can be used to improve the estimate of the arm pose due to the knowledge of strongly the two are related.

Another scenario where knowledge of uncertainties 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.

To introduce knowledge about the state of the system, we include one final error term in the ICP error function:

\[
E_{\text{prior}}(\mathbf{x}) = (\mathbf{x} - \bar{\mu}_k) \bar{\Sigma}_k^{-1}(\mathbf{x} - \bar{\mu}_k)
\] (7)

Although there are numerous reasons, as explained above, to include such an error term in ICP, it should also be noted that adding this prior has the potential to affect the performance of the Kalman filter. \( \bar{\mu}_k \) is supposed to be an independent estimate of the true state, but our prior biases it toward the existing \( \bar{\mu}_k \).

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

\[
E(\mathbf{x}) = E_{\text{pt-plane}}(\mathbf{x}) + \alpha_1 E_{\text{feature}}(\mathbf{x}) + \alpha_2 E_{\text{color}}(\mathbf{x}) + \alpha_3 E_{\text{prior}}(\mathbf{x})
\] (8)
The $\alpha$’s are relative weights for the components of the error function, which we set heuristically.

### 2.3 Handling Multiple Grasps

![Figure 4: Two grasps of the same object. With just a single grasp, the resulting object model will have holes. Between the two grasps, the entirety of the object is visible.](image)

We have so far assumed that modeling begins with the object in the manipulator, and at no time is it let go. The robot’s manipulator will occlude parts of the object as shown in Fig. 4, so to get complete models our algorithm must be able to handle changes in grasp location.

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 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 loosening the constraint on the object pose relative to the manipulator. Finally, in the third state, the expectation is for the object 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.

The first state is the only one in which $\text{Segment and merge object}$, is performed. Even during the grasped state, the modeling procedure is only done when the arm is fully clear of the table.
Figure 5: 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 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 an example of a model built from multiple grasps.

3 OBJECT MODELING

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

3.1 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. 5). 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.

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 $S_m$ or $S_{obj}$ 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 because of the need to maintain explicit connections with other 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 in (Weise et al., 2009), 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. 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 2.2.2), 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.

### 3.2 Model Update

After performing the segmentation described in Section 2, we use surfel update rules similar to Weise et al. (Weise et al., 2009) to modify the object model \( S_{obj} \). 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 sensor reading. \( p_i \) 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 its surrounding pixels are in a single object segment, and \( n_i^* \) does not deviate from the camera direction by more than \( \theta_{max} \).

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 \( \theta_{max} \), 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 $v_i < v_{\text{starve}}$ that has not been seen within the last $t_{\text{starve}}$ 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).

### 3.3 Loop Closure

We also base our loop closure on the techniques developed by Weise et al. (Weise et al., 2009) and refer the interested reader to their work for a more detailed description of the procedure. 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.

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.

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 $S_{C1}, \ldots, S_{CL}$.

Each component cloud is registered into the current frame using ICP, yielding transformations $T^{S}_{C1}, \ldots, T^{S}_{CL}$ bringing the surfels in each component into the sensor frame. The tracking algorithm described in Section 2 implicitly defines a transformation $T^S_{\text{obj}}$ 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 $T^S_{\text{obj}}$ is applied:

$$T_{\text{adj}}^{C_i} = (T^S_{\text{obj}})^{-1} * T^S_{C_i}.$$  

We cannot simply transform each visible surfel by its appropriate $T_{\text{adj}}$ 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 $T_{\text{adj}}$ constraints and relative node pose constraints.

If $T^{\text{init}}_{i}$ and $T^{\text{init}}_{j}$ 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

$$T^{\text{init}}_{j \rightarrow i} = (T^{\text{init}}_{i})^{-1} * T^{\text{init}}_{j}.$$  

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,
Consensus surfel clouds ($S_1, \ldots, S_L$):

1. $S'_1, \ldots, S'_L = S_1, \ldots, S_L$
2. for $i \in \{1, \ldots, L\}$
3. for $j \in \{1, \ldots, |S_i|\}$
4. $depthSum = 0$
5. $normalSum = n_j$
6. $numSurfaces = 1$
7. for $k \in \{1, \ldots, L\}, k \neq i$
8. $S = surfels_in_range(S_k, p_j, r)$
9. if $|S| > 0$
10. $depthSum += \text{avg}\_\text{dist}\_\text{along}\_\text{normal}(S, n_j)$
11. $normalSum += \text{avg}\_\text{normal}(S)$
12. $numSurfaces += 1$
13. $p'_j = p_j + (depthSum/numSurfaces) * n_j$
14. $n'_j = \text{normalize}(normalSum)$
15. return $S'_1, \ldots, S'_L$

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.

To optimize over node poses with the following constraints:

- For each node $i$, if it is in some connected component $C_j$, its pose in the object coordinate frame is constrained to be $T_{adj}^{adj} \cdot T_{i}^{init}$ 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 $T_{i \rightarrow j}^{init}$ 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).

### 3.4 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 $S_{C1}, \ldots, S_{CL}$. 
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. 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 keep around. To avoid throwing away more information than necessary, we use the visibility confidences to guide the removal of surfels. The merging procedure concludes by greedily constructing a new surfel cloud. The surfels from all surfaces are sorted by visibility confidence and are evaluated in order of decreasing confidence. Each surfel 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. 6.

4 RELATED WORK

The existing work in tracking and modeling address subsets of the problem we are trying to solve; however, no one 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.

4.1 Robotics for Object Modeling

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 full surface models but rather sparse sets of oriented 3D points along contours. Another important difference 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 of any widespread use.

Ude et al. (Ude et al., 2008) use robotic manipulation as means of generating 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, this paper assumes neither known camera calibration nor precisely known joint angles. The authors do not make it their goal to construct 3D models and therefore do not need to know 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.

4.2 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 there are no longer encoders to provide (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 would be. 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 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 will fail for objects exhibiting rotational symmetries as many household objects do.

### 4.3 Articulated Tracking

A number of techniques exist for human hand-tracking; however, many of these 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 performing a ray-tracing of 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 low 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 as demonstrated by the wealth of existing ICP variants (see Section 4.4). 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.

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 ICPs search but rather to provide 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.
4.4 ICP Variants

In this paper, we have introduced a number of extensions to ICP, resulting in the error function shown in (8). 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 that descriptors 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 three dimensions.

Other approaches for incorporating additional attributes include matching only a subset of the points which 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 our own 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 as a means of finding 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.

4.5 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 meticulous experimental setup with sophisticated geometric inference algorithms, such as that in the Digital Michelangelo Project (Levoy et al., 2000). In comparison, although we are recovering both object shape and appearance, our objective is not photorealistic rendering, but to robustly and efficiently model objects from an autonomous robot, with an affordable sensor, and to apply such object knowledge in recognition and manipulation.

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
Figure 7: 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’ for display purposes only.

workspace. The specific depth camera we use, developed by PrimeSense (PrimeSense, ), 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, the manipulator grasps and trajectories were specified manually and the objects were grasped only once. Using multiple grasps to generate complete object models is discussed briefly in Section 5.3.

Our current implementation of the algorithm described in Section 2.1 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.

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. 8 an example of the deviation between reported and observed manipulator after 20 seconds of motion at a $2.0^\circ/\sqrt{s}$ noise level. In Fig. 7, 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. 7 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 $R_k$; however, adjusting $R_k$ 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 $3.0^\circ/\sqrt{s}$, while comparable failure levels do not occur until $3.4^\circ/\sqrt{s}$ 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 match. An example model generated for the coffee mug under high noise conditions is shown in Fig. 8. When comparing this model to one built without additional noise (see Fig. 10), 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.
Figure 9: 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. 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 2. 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. 7 to determine how combinations of both modeling and encoder errors affect our tracking algorithm. These results are shown in Fig. 9. The end effector error increases to about 2.5 cm with the shortening of the arm, but our 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.
Figure 10: Shown here are a can and a mug aligned with ICP alone on the left 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 depth scans, resulting in useless object models.

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 align without any information about the hand motion. The resulting clouds are compared with the output of our system in Fig. 10. 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 also note that for objects with color, the presence of the color error term (6) can help improve the object models. The left image in Fig. 11 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 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. 12 illustrates the difference between raw data 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 infor-
Figure 11: (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.

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.

A few of the reconstructed objects are shown in Fig. 13. For rendered videos of the models and videos demonstrating the arm tracking and surfel model construction, see www.youtube.com/ObjectModeling.

5.3 Toward Autonomous Object Modeling

To perform autonomous grasping and modeling, we implemented an approach that 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. We demonstrate this in Fig. 14. While Section 2 details how the tracking
and modeling can be made to work with regrasping, the best way to generate the second grasp is still unclear. Although the first grasp must be made heuristically due to lack of knowledge about the object, the second grasp can be informed by the mostly complete surfel model. The model can be passed along to a grasp planner (e.g. (Berenson and Srinivasa, 2008; Miller and Miller, 2004)) but should be done in a way that discourages grasps covering previously unseen areas. One option is to incorporate into the grasp quality metric an estimate of the area of overlap between the two grasps.

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. 15. 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 objects such as the white mug in Fig. 16, ICP may not always succeed due to the fairly uniform color and geometry.
Figure 13: 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.

Failures such as the two demonstrated here can be resolved by detecting when ICP provides a poor solution and taking appropriate actions. In the first example, the model should not be updated based on the failed alignment. Tracking should proceed without model update until a view is seen sufficient for re-aligning the object model. In the second example, the loop closure should be rejected until a better alignment can be achieved. We are currently exploring metrics indicative of the quality of an alignment.

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 all in one 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 rep-
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 discussed possible techniques for initial and subsequent grasp generation, but these problems remain as future work. There is also the problem of automatic trajectory generation for quick coverage of the object. Finally, by attaching grasp information to our object models and using visual features seen during modeling, a robot could use such models to quickly detect them in the environment using a technique similar to the work of Collet et al. (Collet Romea et al., 2009) and grasp the objects again.

**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.
Figure 15: (left) Poor alignment of the front face of a box after a regrasp due to factors such as high uncertainty in the object pose, non-distinctive geometry, and poor lighting. (right) RGB image from the same frame.

References


Figure 16: The lip of a white mug after a poorly performed loop-closure. ICP was unable to determine the correct transformations of the connected components into the current frame.


