Learning to Segment and Track in RGBD

Alex Teichman, Jake Lussier, and Sebastian Thrun

***Abstract*—We consider the problem of segmenting and tracking deformable objects in color video with depth (RGBD) data available from commodity sensors such as the Asus Xtion Pro Live or Microsoft Kinect. We frame this problem with very few assumptions - no prior object model, no stationary sensor, no prior 3D map - thus making a solution potentially useful for a large number of applications, including semi-supervised learning, 3D model capture, and object recognition.**

Our approach makes use of a rich feature set, including local image appearance, depth discontinuities, optical flow, and surface normals to inform the segmentation decision in a conditional random field model. In contrast to previous work in this field, the proposed method *learns* how to best make use of these features from ground-truth segmented sequences. We provide qualitative and quantitative analyses which demonstrate substantial improvement over the state of the art.

This paper is an extended version of our previous work [25]. Building on this, we show that it is possible to achieve an order of magnitude speedup and thus real-time performance ($\sim$20FPS) on a laptop computer by applying simple algorithmic optimizations to the original work. This speedup comes at only a minor cost in overall accuracy and thus makes this approach applicable to a broader range of tasks. We demonstrate one such task: real-time, online, interactive segmentation to efficiently collect training data for an off-the-shelf object detector.

***Note to Practitioners* — The original motivation for this work derives from object recognition in autonomous driving, where it is desirable to identify objects such as cars and bicyclists in natural street scenes. In previous work [24] it was shown that, so long as one can segment and track objects in advance of knowing what they are, it is possible to train accurate object detectors using a small number of hand-labeled examples combined with a large number of unlabeled examples. The key dependency of a model-free segmentation and tracking method was available because of the structure in the autonomous driving problem. This paper aims to make these techniques applicable in more general environments.**

In the near term, the methods presented here can be used for real-time, online, interactive object segmentation. This can ease the process of collecting training data for existing object recognition systems used in automation today. In the long term, improved implementations could be an integral part of semi-supervised object recognition systems which require few hand-labeled training examples and can produce accurate recognition results.

I. Introduction

The availability of commodity depth sensors such as the Kinect opens the door for a number of new approaches to important problems in robot perception. In this paper, we consider the task of propagating an object segmentation mask through time. It is assumed that a single initial segmentation is given; in this work the initial segmentation is provided by human labeling, but this input could instead come from an automatic method depending on the application. We do not assume the presence of a pre-specified object model (*e.g.* as the Kinect person tracking system assumes a human skeleton), as that would preclude the system from segmenting and tracking arbitrary objects of interest. As such, this task falls into the category of *model-free* segmentation and tracking, *i.e.* no prior class model is assumed. Similarly, we do not assume the sensor is stationary or that a pre-built static environment map is available. As "model-free segmentation and tracking" is somewhat unwieldy, we will sometimes refer to this task as STRGEN (sturgeon), for "segmentation and tracking of generic objects".

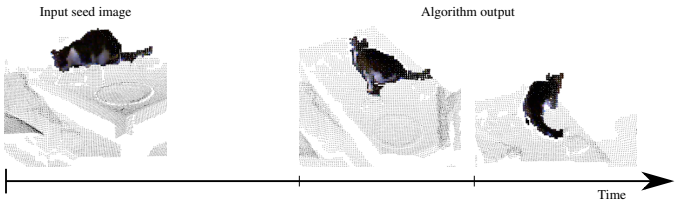

Fig. 1. Definition of the model-free segmentation and tracking task addressed in this paper. Given an initial seed labeling (first frame; cat on a coffee table), the goal is to produce a segmentation mask of the object over time (subsequent frames) without a pre-trained object model. Foreground points are shown bold and colored, background small and gray.

A. Example use cases and long-term vision

There are several reasons a complete solution to the STRGEN task would be useful in robotics, computer vision, and graphics. For example, this would make possible a generic and natural object selection interface which could be used for 3D model capture in unstructured environments. In the following, however, we will focus primarily on the area of object recognition. In the first section we describe how our method can already be a useful part of a standard object detection pipeline; the following sections then outline potential future uses.

1) Training data collection for standard object detectors: Much work has been put into object detection methods using RGBD sensors. While off-the-shelf object detection algorithms such as LINE-MOD [9] are often fast and effective at runtime, training data collection remains a hassle. Typical approaches include turntables and crowdsourcing. While the former allows for high-quality training examples for a full 360° range of views, it also involves an appropriate rig, is limited to objects that can be easily placed in a rig, and must be repeatedly reconfigured for different perspectives. The latter allows for unstructured environments and avoids the physical setup annoyances but also requires a well-designed web application and a data processing framework robust to noisy and incompatible responses. This approach also could be inapplicable to certain data sources due to privacy concerns.

Fig. 2. Example usage of model-free segmentation and tracking algorithm for object detector training set collection. Initially, no object is being tracked. A user provides sparse foreground and background hints as the RGBD data streams in. Segmented objects are shown in red.

Other data labeling approaches, such as iteratively placing objects in a scene and subtracting a known background, are rare as they often have disadvantages such as being slower to collect significant quantities of training data.

Using a STRGEN algorithm for the collection of training data avoids these disadvantages. With a handheld depth sensor and laptop computer, training data can be collected in unstructured environments by taking sparse labeling hints from the user and generating segmentations over time, providing new training examples with different object poses, view angles, lighting conditions, and so on, with little additional human effort. Labeled, segmented objects can be collected in real time while the raw sensor data is recorded. A prototype of such a system is shown in Figure 2; see also Section IV-C for an experiment using a similar concept.

2) Track classification: Currently, most object recognition methods fall into semantic segmentation or sliding window categories. When a solution to the STRGEN problem is available, however, it is possible to use a different approach, exemplified in [23]. In this approach, objects are segmented and tracked over time without a semantic label and the track as a whole is classified online as new segmented frames stream in; we refer to this object recognition problem breakdown as STAC, for segmentation, tracking, and classification. As shown in [23], the STAC approach significantly improves classification accuracy because of its ability to use multiple views over time and to use track descriptors such as average object speed or maximum angular velocity. In [23], model-free segmentation and tracking was available due to the specific circumstances of autonomous driving that are not generally applicable. A solution to the general model-free segmentation and tracking problem, however, would enable this approach in less structured environments.

3) Tracking-based semi-supervised learning: A solution to the STAC problem opens the door to a simple and effective method of semi-supervised learning in which a large number of unlabeled tracks of objects are used in conjunction with a small number of hand-labeled tracks to learn an accurate classifier. This method, known as tracking-based semi-supervised learning (TBSSL), was demonstrated in the autonomous driving context using laser range finders to learn to accurately recognize pedestrians, bicyclists, and cars versus other distractor objects in the environment using very few hand-labeled training examples [24]. However, as with [23], model-free segmentation and tracking was more or less freely available because objects on streets generally avoid collision and thus remain depth-segmentable using simple algorithms. To extend tracking-based semi-supervised learning to the more general case in which objects are not easily depth-segmentable, a more general solution to the STRGEN problem is required.

4) Long-term vision: The primary long-term motivation for this work is to enable object recognition systems that produce high-quality results, yet are trainable and usable by regular people using only modest amounts of effort. The success of STAC and TBSSL, given model-free segmentation and tracking, hint that this may be possible. One option is to (A) collect a small number of training examples using a natural interface such as that of Figure 2, (B) apply TBSSL to learn a high-quality classifier from large quantities of unlabeled data, and (C) run STAC to recognize objects using the learned classifier. The primary missing component in this system or variations thereof is a high-accuracy and real-time solution to the general STRGEN problem.

II. Previous work

While there is much previous work on tracking in general, our problem's online nature, output of segmentation masks rather than bounding boxes, and lack of simplifying assumptions restrict the directly-related previous work substantially. The most similar and recent work to ours is HoughTrack [7], which uses Hough Forests and GrabCut to segment consecutive frames. We provide a quantitative comparison to HoughTrack in Section IV-A. Older previous work on this problem include [3] and their multi-object extension [4], which use a generative model and level sets; [12], which tracks local object patches using Basin Hopping Monte Carlo sampling; and [16], which uses a conditional random field model and loopy belief propagation. All these works limit themselves to only a small number of features and do not use learning methods to find the best general-purpose tracker. Additionally, none of these works consider depth information.

We now briefly review previous work on related problems.

Interactive segmentation - Human-assisted object segmentation has been the subject of much work including interactive graph cuts [5], GrabCut [17], and Video SnapCut [2]. These methods are largely intended for graphics or assisted-labeling applications as they require a human in the loop. This paper addresses automatic segmentation with the exception of the application example in Section IV-C.

Bounding box tracking - Discriminative tracking [8], [20], [11] addresses the online, model-free tracking task, but where the goal is to track arbitrary objects in a bounding box rather than provide a segmentation mask.

Rigid object tracking - Rigid object tracking using depth information, such as the open source method in PCL [18], addresses a similar but simpler problem, as it is not designed to work with deformable objects.

Offline methods - The work of [6] takes as input segmentations for the first and last frames in a sequence, rather than just

the first frame. While [26] takes as input only the segmentation of the first frame, they construct a CRF on the entire video sequence, with CRF labels corresponding to object flows. Here we are interested in a method that can update as new data streams in, thus making it applicable for robotics or other online vision tasks.

Background subtraction - Background subtraction approaches can greatly simplify the segmentation task. For example, [1] assumes a stationary camera to produce fine-grained segmentation masks for multiple objects. With depth sensors, background subtraction methods can also operate while the sensor is in motion: Google's autonomous car project uses pre-built 3D static environment maps to subtract away all 3D points except the moving obstacles [28]. This enables simple depth segmentation to be effective as long as moving objects do not touch, but assumes a high-precision localization system as well as a pre-learned static environment map.

Model-based tracking - For some tasks, it is appropriate to model a specific object that is to be tracked, either with an explicit 3D model as in [15] or with pre-trained statistical models that are specific to the particular object class. Examples of the latter approach include the human pose tracking of the Kinect [19] and model-based car tracking in laser data for autonomous driving [14].

It is worth noting that the line between model-free and model-based tracking is not sharp. Because several of our intended applications revolve around training detectors for previously-unseen objects, we restrict ourselves to making use of object models that are learned on the fly. It is entirely conceivable, however, that one could plug in a detailed pre-trained object model to our framework to produce higher accuracy results for that particular object.

Similarly, using a SLAM system to generate a known background map would improve the segmentation process by allowing for static background subtraction even with a moving sensor. However, a SLAM system would prevent the segmentation algorithm from working in unmapped areas or on objects that are part of the static environment, so we consider only background models that are learned on the fly and are relative to the chosen foreground object.

III. Approach

There are a large number of possible cues that could inform a segmentation and tracking algorithm. Optical flow, image appearance, 3D structure, depth discontinuities, color discontinuities, etc., all provide potentially useful information. For example:

- An optical flow vector implies that the label of the source pixel at time $t-1$ is likely to propagate to that of the destination pixel at time t.
- Pixels with similar color and texture to previous foreground examples are likely to be foreground.
- The shape of the object in the previous frame is likely to be similar to the shape of the object in the next frame.
- Points nearby in 3D space are likely to share the same label.
- Nearby points with similar colors are likely to share the same label.

Previous work generally focuses on a few particular features; here, we advocate the use of a large number of features. The above intuitions can be readily encoded by node and edge potentials in a conditional random field model. However, this additional complexity introduces a new problem: how much importance should each feature be assigned? While using a small number of features permits their weights to be selected by hand or tested with cross validation, this quickly becomes impractical as the number of features increases.

The margin-maximizing approach of structural SVMs [22], [27], adapted to use graph cuts for vision in [21], provides a solution to learning in these scenarios. Intuitively, this method entails running MAP inference, and then adding constraints to an optimization problem which assert that the margin between the ground truth labeling and the generated (and generally incorrect) labeling should be as large as possible. The application of this approach will be discussed in detail in Section III-B.

A. Conditional random fields and inference

A conditional random field is an undirected graphical model that can make use of rich feature sets and produce locally-consistent predictions. It aims to spend modeling power on only the distribution of the target variables given the observed variables. In particular, the conditional random field takes the form

$$\mathbb{P}(y|x) = \frac{1}{Z(x)} \exp(-E(y,x)), \tag{1}$$

where Z is the normalizer or partition function, $y \in \{-1,+1\}^n$ is the segmentation for an image with n pixels, and x is a set of features defined for the RGBD frame, to be detailed later. The energy function E contains the features that encode various intuitions about what labels individual pixels should take and which pixels should share the same labels. In particular, the energy function is defined as

$$E(y,x) = \sum_{i \in \Phi_v} w_i \sum_{j \in v_i} \phi_j^{(i)}(y,x) + \sum_{i \in \Phi_{\mathscr{E}}} w_i \sum_{(j,k) \in \mathscr{N}_i} \phi_{jk}^{(i)}(y,x). \tag{2}$$

Here, Φ_v is the set of node potential indices (*i.e.* one for each type of node potential such as local image appearance or 3D structure alignment), v_i is the set of all node indices for this potential type (normally one per pixel), and $\phi_j^{(i)}$ is the node potential of type i at pixel j. Similarly, $\Phi_{\mathscr{E}}$ is the set of edge potential indices, $\mathscr{N}_i$ is the neighborhood system for edge potential type i (normally pairs of neighboring pixels), and $\phi_{jk}^{(i)}$ is the edge potential between pixels j and k for edge potentials of type i. Thus, the weights apply at the feature-type level, *i.e.* w_i describes how important the ith feature is.

The goal of MAP inference in this model is to choose the most likely segmentation y given the features x by solving

$$\underset{y}{\text{maximize}} \quad \mathbb{P}(y|x) = \underset{y}{\text{minimize}} \quad E(y,x). \tag{3}$$

During inference, the weights w remain fixed. The solution to this problem can be efficiently computed for $y \in \{-1,+1\}^n$ and submodular energy function E using graph cuts [5].

$$\begin{aligned} \underset{w,\xi}{\text{minimize}} \quad & \frac{1}{2}||w||^2 + \frac{C}{M}\sum_{m=1}^{M}\xi_m \\ \text{subject to} \quad & \xi \geq 0 \\ & E(y,x_m) - E(y_m,x_m) \geq \Delta(y_m,y) - \xi_m \quad \forall m, \forall y \in \mathscr{Y} \end{aligned} \tag{4}$$

$$\begin{aligned} \underset{w,\xi}{\text{minimize}} \quad & \frac{1}{2}||w||^2 + C\xi \\ \text{subject to} \quad & \xi \geq 0 \\ & \frac{1}{M}\sum_{m=1}^{M} E(\hat{y}_m,x_m) - E(y_m,x_m) \geq \frac{1}{M}\sum_{m=1}^{M}\Delta(y_m,\hat{y}_m) - \xi \quad \forall (\hat{y}_1,...,\hat{y}_M) \in \mathscr{Y}^M \end{aligned} \tag{5}$$

B. Learning

Given a dataset of (y_m, x_m) for $m = 1 \ldots M$, the goal of CRF learning is to choose the weights w that will result in the lowest test error. While it would be desirable to learn the weights directly using the maximum likelihood approach

$$\underset{w}{\text{maximize}} \quad \prod_m \mathbb{P}(y_m|x_m), \tag{6}$$

this is not possible to do exactly because of the presence of the partition function $Z(x) = \sum_y \exp(-E(y,x))$ in the gradient. This function sums over all 2^n segmentations of an image with n pixels.

Fortunately, there is an alternative approach known as the structural support vector machine [22], [27], [21], which we now briefly review. Solving the margin maximizing optimization problem of (4) would result in a good solution. Here, C is a constant, Δ is a loss function, ξ_m is a slack variable for training example m, and $\mathscr{Y}$ is the set of all possible labelings of an image. This problem, too, is intractable because it has exponentially many constraints. However, one can iteratively build a small, greedy approximation to the exponential set of constraints such that the resulting weights w are good.

Further, one can transform (4), known as the n-slack formulation, into the equivalent problem (5) as described in [10]. As before, while the number of constraints is exponential, one can build a small, greedy approximation that produces good results. Known as the 1-slack formulation, this problem is equivalent and can be solved much more efficiently than (4), often by one or two orders of magnitude.

The goal is to learn the weights that will best segment an entire sequence given a single seed frame. However, the segmenter operates on a single frame at a time, so our training dataset must be in the form of (y,x) pairs. As will be discussed in Section III-C, some of the features are stateful, *i.e.* they depend on previous segmentations. This presents a minor complication. At training time, we do not have access to a good choice of weights w, but the features depend on previous segmentations which in turn depend on the weights. To resolve this, we adopt the simple strategy of generating features assuming that previous segmentations were equal to ground truth. This dataset generation process is specified in Algorithm 1.

The structural SVM solver is detailed in Algorithm 2. Because the graph cuts solver assumes a submodular energy function, non-negative edge weights w_i for all $i \in \Phi_{\mathscr{E}}$ are required. As our node potentials are generally designed to produce values with the desired sign, we constrain them in Algorithm 2 to have non-negative weights as well. The term $\Delta(y_m, y)$ is 0-1 loss, but a margin rescaling approach could easily be obtained by changing Δ to Hamming loss and making a small modification to the graph cuts solver during learning (see [21]).

Algorithm 1 Training set generation

```
𝕊 = {S : S = ((y_0, d_0), (y_1, d_1), ...)} is a set of sequences,
      where y_i is a ground truth segmentation and d_i
      is an RGBD frame
𝒟 = ∅
for S ∈ 𝕊 do
  Initialize stateful components, e.g. the patch classifier that
      learns its model online
  for (y_i, d_i) ∈ S do
    Update stateful components using y_{i-1} as the previous
        segmentation
    Generate features x
    𝒟 := 𝒟 ∪ {(y_i, x)}
  end for
end for
return 𝒟
```

C. Energy function terms

We now review the particular choice of energy function (2) that we use in our implementation. The particular choice of features can be modified or added to without substantially changing the underlying framework. See Figure 3 for visualizations. All node potentials are designed to be constrained to $[-1,+1]$, and all edge potentials to $[-1,0]$. While not strictly necessary, this aids in interpreting the weights learned by the structural SVM.

Algorithm 2 Structural SVM for learning to segment and track

$\mathscr{D}$ is a set of training examples (y,x), formed as described in Algorithm 1.
C and ε are constants, chosen by cross validation.

$\mathscr{W} \leftarrow \emptyset$
repeat
Update the parameters w to maximize the margin.

$$\begin{aligned} \underset{w,\xi}{\text{minimize}} \quad & \frac{1}{2}||w||^2 + C\xi \\ \text{subject to} \quad & w \geq 0, \quad \xi \geq 0 \\ & \frac{1}{M}\sum_{m=1}^{M} E(\hat{y}_m, x_m) - E(y_m, x_m) \geq \frac{1}{M}\sum_{m=1}^{M} \Delta(y_m, \hat{y}_m) - \xi \quad \forall (\hat{y}_1, ..., \hat{y}_M) \in \mathscr{W} \end{aligned}$$

for $(y_m, x_m) \in \mathscr{D}$ **do**
Find the MAP assignment using graph cuts.
$\hat{y}_m \leftarrow \text{argmin}_y E(y, x_m)$
end for
$\mathscr{W} \leftarrow \mathscr{W} \cup \{(\hat{y}_1, \ldots, \hat{y}_M)\}$
until $\frac{1}{M}\sum_{m=1}^{M} \Delta(y_m, \hat{y}_m) - E(\hat{y}_m, x_m) + E(y_m, x_m) \leq \xi + \varepsilon$

1) Node potentials: Node potentials capture several aspects of shape, appearance, and motion. All take the form

$$\phi_j(y,x) = \begin{cases} a_j & \text{if } y_j = 1 \\ b_j & \text{if } y_j = -1, \end{cases} \tag{7}$$

where a_j and b_j are functions of the data x, and $y_j = 1$ indicates that pixel j has been assigned to the foreground.

Seed labels - The first frame of every sequence provides a foreground/background segmentation. Seed label node potentials can also be used after the first frame as additional segmentation hints, used in this paper only for construction of the ground truth dataset and for the interactive experiment of Section IV-C. Each pixel in the image can be labeled by the user as foreground or background, or left unlabeled. Concretely, let $s_j \in \{-1, 0, +1\}$ be the seed labels for each pixel. Then, the potential is defined by

$$\phi_j(y,x) = \begin{cases} -1 & \text{if } y_j = s_j \\ 0 & \text{otherwise.} \end{cases}$$

Optical flow - Pixels in the current frame with optical flow vectors from the previous frame are likely to have the label of the originating pixel. If no optical flow vector terminates at pixel j in the current frame, let $f_j = 0$. Otherwise, let f_j be equal to the label of the originating pixel from the previous frame. Then, the potential is defined by

$$\phi_j(y,x) = \begin{cases} -1 & \text{if } y_j = f_j \\ 0 & \text{otherwise.} \end{cases}$$

Frame alignment bilateral filter - Optical flow provides correspondences from one frame to the next; after aligning these frames, a bilateral filter given 3D position and RGB values is used to blur the labels from the previous frame into the current frame, respecting color and depth boundaries.

Specifically, alignment between the two frames is found using a RANSAC method, with 3D correspondences given by optical flow vectors. Flow vectors that originate or terminate on a depth edge are rejected, as their 3D location is often unstable. After alignment, we find the set of neighboring points N from the previous frame within a given radius of the query point, then compute the value

$$z_j = \sum_{k \in N} y'_k \exp\left(-\frac{||c_j - c_k||_2}{\sigma_c} - \frac{||p_j - p_k||_2}{\sigma_d} \right),$$

where y'_k is the $\{-1, +1\}$ label of point k in the previous segmentation, c_j is the RGB value of pixel j, and p_j is the 3D location of pixel j. The two σs are bandwidth parameters, chosen by hand. The value z_j is essentially that computed by a bilateral filter, but without the normalization. This lack of normalization prevents a point with just a few distant neighbors from being assigned an energy just as strong as a point with many close neighbors. Referring to (7), the final energy assignment is made by setting $b_j = 0$ and $a_j = 1 - 2/(1 + e^{-z_j})$.

Patch classifiers - Two different parameterizations of a random fern patch classifier similar to [13] are trained online and used to make pixel-wise predictions based on local color and intensity information.

A random fern classifier is a semi-naive Bayes method in which random sets of bit features are chosen ("ferns"), and each possible assignment to the bit features in the fern defines a bin. Each of these bins maintains a probability estimate by simple counting. The final probability estimate is arrived at by probabilistically combining the outputs of many different randomly generated ferns. See [13] for details. We use 50 randomly generated ferns, each of which have 12 randomly generated features.

Each bit feature in a fern is generated by very simple functions of the data contained in the image patch. For example, one of the bit features we use is defined by

$$z = \begin{cases} 1 & \text{if } I_{p_0} < I_{p_1} \\ 0 & \text{otherwise,} \end{cases}$$

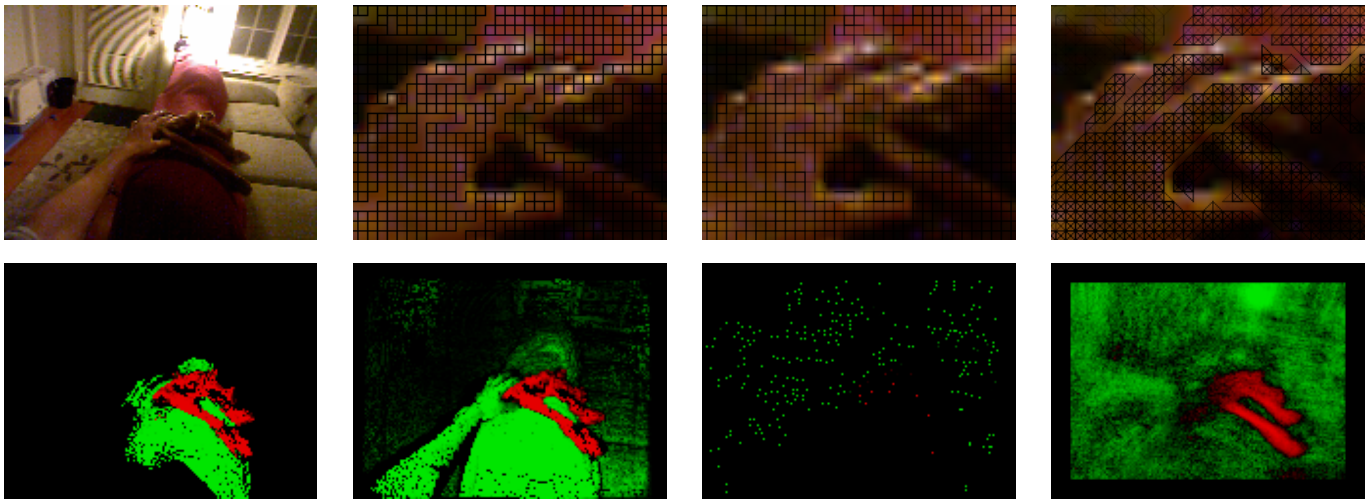

Fig. 3. Visualizations of selected edge and node potentials in the CRF. Edge potentials are zoomed in to show fine structure. Strong edge potentials are darker, while weaker edge potentials fade to original image color. Node potentials expressing a preference for foreground are shown in red, for background in green, and for neither in black. Top row: original image, canny edge potentials, color distance, depth edge potentials. Bottom row: ICP, frame alignment bilateral filter, optical flow, and patch classifier node potentials. Best viewed in color.

where p_0 and p_1 are two pre-determined points in the image patch, and I_p is the intensity of the camera image at point p. Other bit features include color comparisons and Haar wavelet comparisons, efficiently computed using the integral image.

At each new frame, the classifier is updated using the graph cuts segmentation output from the previous frame as training data, then is used to compute a probability of foreground estimate for every pixel in the current frame. The value a_j is set to the probability of background, and b_j to probability of foreground.

Distance from previous foreground - Points very far away from the object's position in the previous frame are unlikely to be foreground.

After alignment of the current frame to the previous frame using the same optical-flow-driven method as the bilateral term, we compute the distance d from the point j in the current frame to the closest foreground point in the previous frame. The final potentials are set to $a_j = 0$ and $b_j = \exp(-d/\sigma) - 1$.

ICP - The iterative closest point algorithm is used to fit the foreground object's 3D points (and nearby background points) from the previous frame to the current frame. If the alignment has a sufficient number of inliers, points in the current frame are given node potentials similar to those of the bilateral node potential, but where the ICP-fit points are used instead of the entire previous frame.

Prior term - To express a prior on background, we add a term for which $a_j = 0$ and $b_j = -1$.

2) Edge potentials: Edge potentials capture the intuition that the foreground/background boundary is likely to be at an image gradient, color change, depth discontinuity, or surface normal change. All take the form

$$\phi_{jk}(y,x) = \begin{cases} a_{jk} & \text{if } y_j = y_k \\ 0 & \text{otherwise,} \end{cases}$$

where a_{jk} is a function of the data x. All edges are between neighboring points in the image.

Canny edges - All neighboring pixels are connected by an edge potential except those cut by canny edges. Concretely,

$$a_{jk} = \begin{cases} -1 & \text{if neither pixel lies on a Canny edge} \\ 0 & \text{otherwise.} \end{cases}$$

Color distance - Edge weights are assigned based on Euclidean distance between neighboring RGB values; $a_{jk} = -\exp(-||c_j - c_k||/\sigma)$, where c_j and c_k are the RGB values of the two pixels and σ is a bandwidth parameter, chosen by hand.

3D distance - Points nearby in 3D are likely to share the same label, especially if they lie in the same plane. Out-of-plane distance changes are penalized more heavily than in-plane distance changes. Specifically,

$$a_{jk} = -\exp\left(-\frac{|(p_j - p_k)^T n_k|}{\sigma_n} - \frac{||p_j - p_k||_2}{\sigma_d}\right),$$

where p_j and p_k are the neighboring 3D points, n_k is the surface normal at point p_k, and the σs are bandwidth parameters chosen by hand.

Surface normals - Neighboring points that share the same surface normal are likely to have the same label. See Figure 4. We use $a_{jk} = -\exp(-\theta/\sigma)$, where θ is the angle between the two normals and σ is a bandwidth parameter chosen by hand.

Edge potential products - Several combinations of the above are also provided, taking the form $a_{jk} = -|\prod_i a_{jk}^{(i)}|$, where $a_{jk}^{(i)}$ is the value from one of the above edge potentials. Intuitively, this encodes an edge potential that is strong (*i.e.* favors label equality) if all of the component edge potentials are strong.

IV. Experiments

We provide a quantitative analysis to demonstrate improvement over the state of the art and a qualitative discussion of the strengths and weaknesses of the current implementation. All experiments use QQVGA (*i.e.* 160×120) RGBD data.

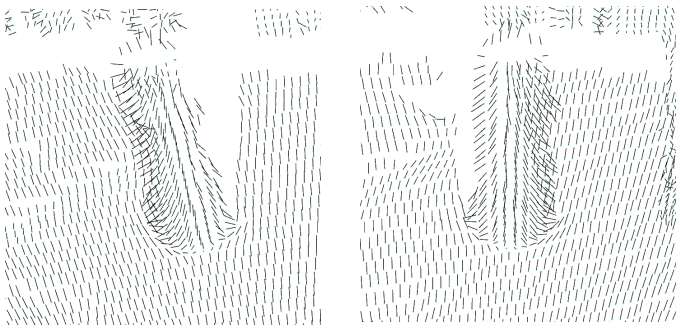

Fig. 4. While image appearance and depth information may sometimes provide little information about an object boundary, a change in surface normals can be informative. Surface normals are shown as small dark lines, seen from two angles to aid in depth perception.

The structural SVM of Algorithm 2 was implemented with standard interior point solver techniques. The graph cuts solver of [5] was used for inference.

A. Quantitative analysis

As there is no work we are aware of which does segmentation and tracking of non-rigid objects using depth data or learning of the best general purpose segmenter, we compare against HoughTrack, discussed in Section II. We used the implementation that accompanies [7]. To ensure a fair comparison, we modified the HoughTrack implementation to be initialized with a segmentation mask rather than a bounding box.

No RGBD segmentation and tracking dataset currently exists, so we generated one containing 28 fully-labeled sequences with a total of about 4000 frames. Objects include, for example, a sheet of paper on a desk, cat, jacket, mug, laptop, and hat. In general, the dataset includes objects with different levels of rigidity, texture, and depth change from the surroundings. Ground truth was generated by hand labeling, assisted with an interactive version of the segmenter, similar to the interactive graph cuts work of [5].

The dataset was split into a training set of about 500 frames over 10 sequences and a testing set of about 3500 frames over 18 sequences. Training of our method was run once on the training set and the resulting segmenter was used to produce results for all testing sequences. HoughTrack has no training stage, and thus did not use the training set. Testing results were produced for both by providing an initial segmentation for the first frame of each sequence, and all further frames were segmented without additional input.

Individual frames are evaluated with two different metrics. Hamming loss, or total number of pixels wrong, is simple and direct but ignores the size of the object; a Hamming loss of twenty could be negligible or a significant fraction of the object in question. To correct for this, we also report *normalized accuracy*, which is 1 if all pixels in the frame are correctly labeled and 0 if the number of incorrectly labeled pixels equals or exceeds the number of pixels in the foreground object. More precisely, normalized accuracy is $1 - \min(1, \texttt{num\_wrong}/\texttt{num\_fg})$. Sequence results are reported as the average of these values over all frames.

Overall, our method demonstrates a normalized error reduction of about 65% compared to the state-of-the-art. Detailed results can be seen in Figure 5. Training of our method takes a total of about 10 minutes and is done only once. At runtime, HoughTrack's implementation takes about 200ms per frame, whereas our method's implementation takes about 1200ms per frame.[1] Fortunately, it is possible to dramatically speed up our method; see Section IV-B3.

B. Qualitative analysis

1) Strengths: Our results demonstrate that this method can be effective even in sequences which include significant non-rigid object transformations, occlusion, and a lack of visually distinguishing appearance. As seen in the illustration in Figure 3, depth provides a very powerful cue as to what neighboring points are likely to share the same label. In addition to depth discontinuities, surface normals (Figure 4) can provide useful information about object boundaries, enabling the segmentation and tracking of objects for which using visual appearance alone would be extremely challenging, such as that of Figure 6. While this sequence would be relatively easy for off-the-shelf 3D rigid object trackers such as that in PCL [18], the tracking of deformable objects such as cats and humans would not be possible.

Our approach is general enough to handle both objects with few visually distinguishing characteristics and objects which are non-rigid; see Figure 7 for examples. In particular, the interface between the cat and the girl in sequences (A) and (B), and the interface between the cat and the bag in sequence (C) are maintained correctly. The method also works for objects without significant depth, such as the paper on the desk in sequence (G), and can recover from heavy occlusion as in sequence (F). The hat in sequence (H) undergoes deformation and heavy RGB blurring due to darkness of the environment, yet is tracked correctly. Sequence (D) shows the tracking of a piece of bread with Nutella while it is being eaten.

2) Weaknesses and future work: As is typical in tracking tasks, stability versus permissiveness is a tradeoff. In some cases, it is not well defined whether a recently-unoccluded, disconnected set of points near the foreground object should be part of the foreground or part of the background; this is a common cause of errors in the current implementation. An example of this occurs in sequence (E), when the cat's head becomes self-occluded, then unoccluded, and the system cannot determine that these points should be assigned to foreground. The error is then propagated. Similarly, the girl's far arm in sequence (A) is lost after being occluded and then unoccluded. Rapid motion appears to exacerbate this problem. In general, it appears that more work is required to produce acceptable results on objects such as entire people which have many rapidly self-occluding and self-unoccluding articulated parts. While partial occlusion can be handled, as in sequence (F), the current implementation has no facility for re-acquisition if the target is completely lost. Finally, thin

[1] Both implementations were compiled with optimizations turned on, run on the same machine, and given one thread.

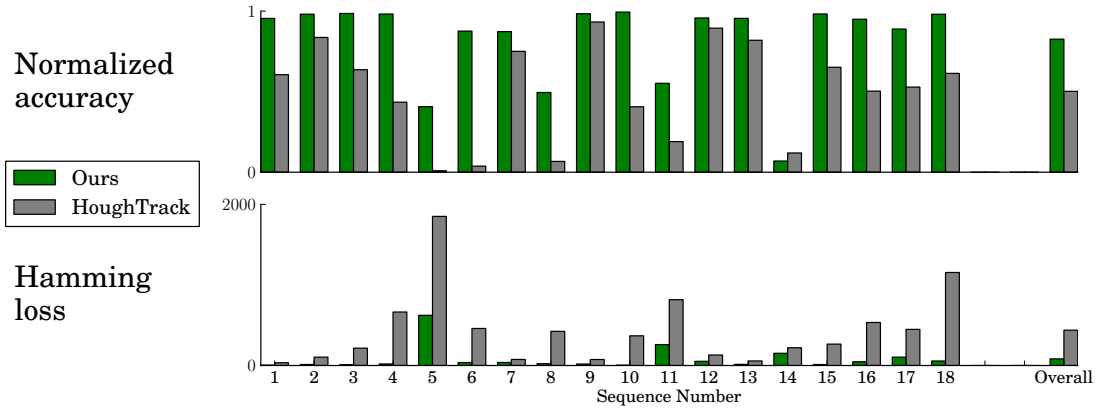

Fig. 5. Comparison of our method with the most similar state-of-the-art work. Hamming loss is absolute number of pixels wrong; normalized accuracy takes into account object size and has a maximum of one. Total error reduction is about 65%.

Fig. 6. Visualization of results in which image appearance alone would lead to a very difficult segmentation problem, but which becomes relatively easy when reasoning about depth as well.

parts of objects are often problematic, as it seems the edge potentials are not well connected enough in these areas.

Fortunately, it is likely that improvements to node and edge potentials can resolve these limitations. Using longer range edges (rather than a simple 2D grid connecting neighbors only) could improve performance on thin objects and quickly moving objects, and improvements in the image-appearance-based patch classifier could result in better segmentations of objects that self-occlude parts frequently. A node potential based on LINE-MOD or a discriminative tracker such as [11] could solve the re-acquisition problem, and could be run on object parts rather than the object as a whole to enable usage on non-rigid objects. This could be particularly helpful for parts of objects like arms which become briefly occluded and then unoccluded, but are disconnected in the 3D pointcloud from the rest of the person.

The feature-rich representation we use presents a challenge and an opportunity. There are a large number of parameters in the computation pipeline which cannot be learned via the structural SVM (*e.g.* the σ's discussed in Section III-C). There is also substantial freedom in the structure of the computation pipeline. Choosing the structure and the parameters by hand (as we have done here) is possible, but onerous; there is an opportunity to learn these in a way that maximizes accuracy while respecting timing constraints.

Finally, the method we have presented learns how to combine a diverse set of features for the best possible general purpose segmenter. There is a significant opportunity to make this method learn *on the fly* the best weighting of the features for a particular object. For example, it could be learned online that the surface normal edge potentials are not very helpful while tracking a poster on the wall.

3) Real-time performance: So far, we have discussed a relatively slow implementation of the segmentation and tracking algorithm. While this version is still useful for tasks such as offline object labeling or model building, a real-time algorithm

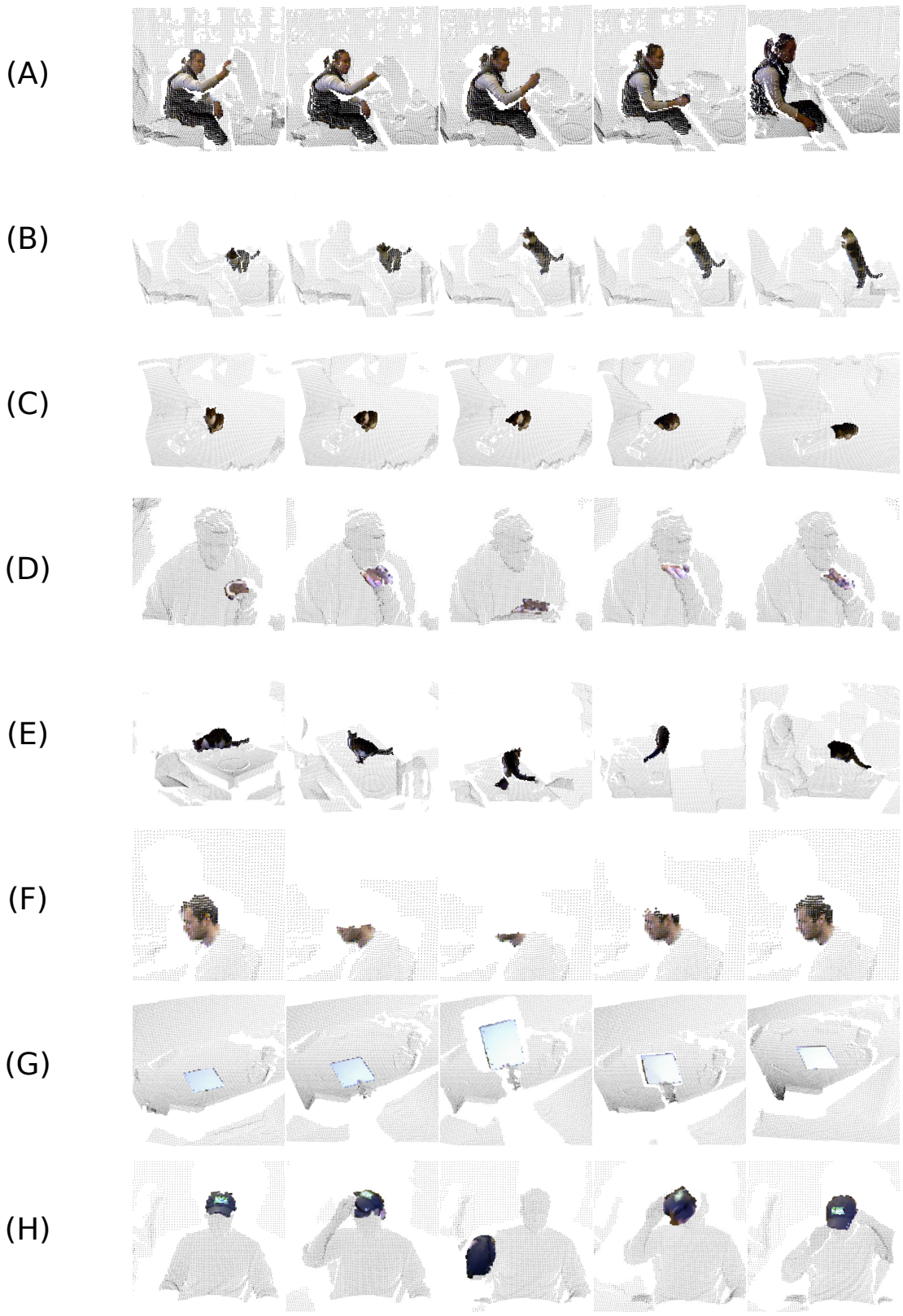

Fig. 7. Visualization of results. The first frame in each sequence (far left) is the seed frame. Foreground points are shown in bold and color while background points are shown small and gray. Best viewed on-screen.

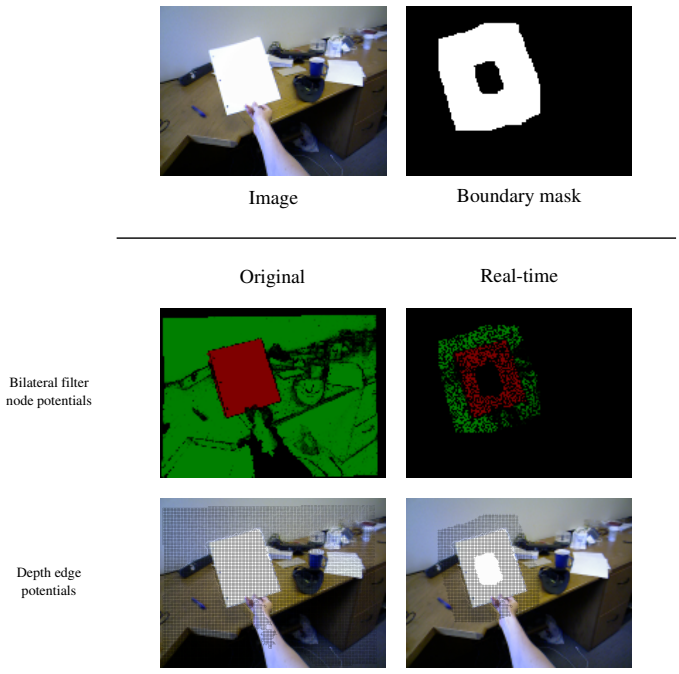

Fig. 8. Visualization of the original method versus the real-time version. Node potential and edge potential visualizations follow the same conventions as in Figure 3. Best viewed in color.

is desirable. This is particularly important in realizing some of the goals discussed in Section I-A, such as using the algorithm as a component of tracking-based semi-supervised learning. In this section, we show how to achieve real-time performance with only a very small loss in segmentation accuracy. Feature computation is by far the bottleneck, so we can focus our efforts entirely on optimizing this part of the pipeline.

We make use of two key ideas in achieving real-time performance. First, one need only compute features and feature pre-requisites at the boundaries of the target object. Intuitively, there is no point in computing surface normals in the upper right corner of the RGBD image when the object of interest is in the lower left. Second, it is not necessary to compute expensive node potentials at every pixel in the image because the edge potentials in the CRF will tend to smooth out the final segmentation. Examples comparing the original and real-time methods can be seen in Figure 8.

Concretely, at each timestep, a boundary mask is computed by dilating the background and foreground masks from the previous segmentation and taking their intersection; this provides a mask that highlights only the edges of the object. Then, all following steps in the computation pipeline compute only inside the boundary mask. Node and edge potentials described in Section III-C are thus not computed in non-boundary regions. Instead, we force all of these pixels to take the label they had from the previous segmentation. Further, inside the boundary mask, downsampling is applied to the patch classifier, bilateral, and ICP node potentials.

To show that this optimization does not adversely affect segmentation performance, we ran ten iterations comparing the optimized segmentation algorithm versus the original algorithm using the same experimental setup as Section IV-A. Quantitative results showing overall segmentation accuracy and runtimes are shown in Figure 9. Experiments were run using a multi-threaded implementation on a laptop with an i7-3820QM processor. With use of the boundary mask and 50% downsampling, it is possible to run the algorithm at $\sim$20FPS without sacrificing too much accuracy.

These results were all computed using QQVGA images. While this may seem unreasonably small, it is actually acceptable for tasks that do not require high resolution boundaries. For example, you might want to use the segmentation and tracking algorithm to collect training data for an object detector. The detector might require VGA (*i.e.* 640×480) data to produce good results, but computing the segmentation at QQVGA and scaling it up to the VGA image is acceptable.

Even at VGA, it is conceivable that real-time operation could be obtained with significant improvement to the feature set and code optimizations. As shown in Figure 10, the graph cuts solver does not preclude real-time segmentation.

C. Use-case example: training object detectors

In this section, we describe the application of our segmentation and tracking method to the problem of data labeling in supervised object detection systems such as LINE-MOD. Such systems rely on well-segmented training data typically obtained from either meticulously rigged turntables or elaborate crowd-sourcing applications, as discussed in Section I-A1. In contrast, by slightly modifying the STRGEN algorithm to incorporate the user's segmentation hints as node potentials (discussed in Section III-C1) throughout the sequence, we provide an interactive, real-time means to accurately collect segmented training examples with minimal environmental control. Concretely, the original algorithm generated a node potential for seed labels on the first frame; here we simply allow these node potentials to be generated for subsequent frames as well.

The process, illustrated in Figure 11, only assumes the user is equipped with a laptop and an RGBD depth sensor. For our experiments, we attached an Asus Xtion Pro Live to the laptop of Section IV-B3. To begin, the user starts recording the RGBD sequence and provides some rough initial foreground and background labels. These are interpreted as node potentials by the STRGEN algorithm and the segmentation is propagated as additional frames stream in. In the case of errant segmentations, the user provides in real time additional foreground and background labels, and the STRGEN algorithm incorporates these as with the initial hints. At the end of the process, the user is left with a collection of segmented objects instances. Not all segmentations are perfect, but almost all bad segmentations are followed by a user correction. Thus, we use the five second rule: All segmentations that were generated less than about five seconds before any user-generated segmentation hint are ignored and the rest are added to the training set. At this point, it would probably be beneficial to generate many artificial viewpoint variations and add these to the training set, as well; this step is bypassed for this small experiment.

In Figure 12, we demonstrate how these segmented examples can be used to construct an object detector. We offer

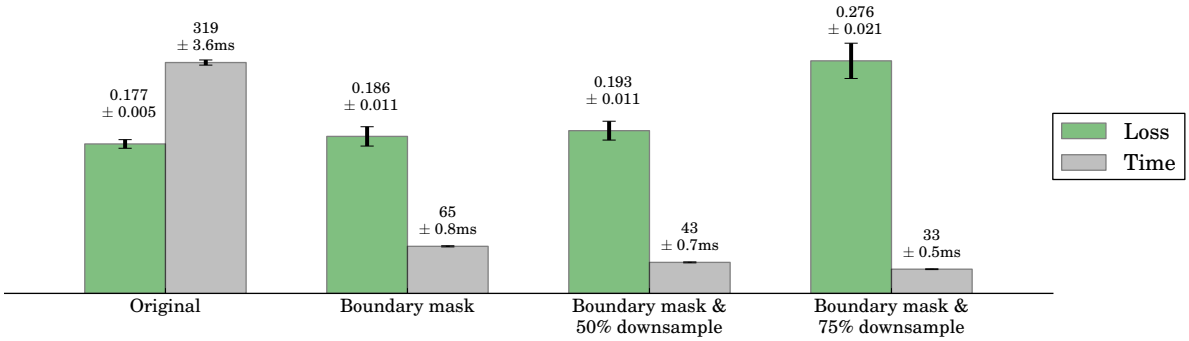

Fig. 9. Test set normalized loss and average per-frame segmentation time compared with different levels of algorithmic optimization as discussed in Section IV-B3. Thirteen runs of each condition were made; error bars and annotations show one standard deviation for this dataset. While a 75% downsampling induces too much error to be acceptable, 50% downsampling appears to be a good trade-off for many applications. Timing results for the original method differ from those presented in Section IV-A because multithreading is enabled here.

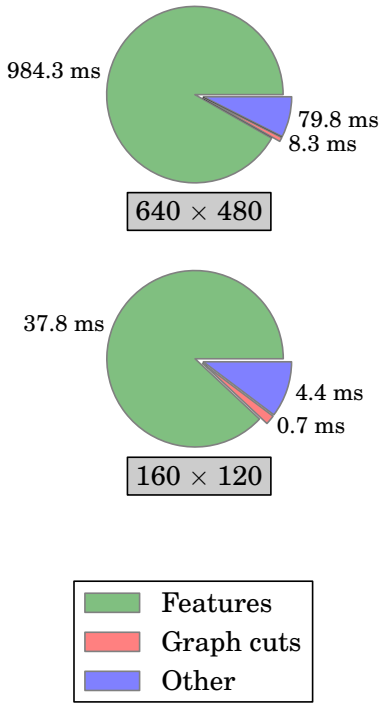

Fig. 10. Average frame segmentation timing breakdown for VGA and QQVGA resolutions using the boundary mask and 50% downsampling of node potentials. Feature computation dominates and the graph cuts solver does not preclude real-time operation even at VGA resolution; thus, real-time segmentation at higher resolutions is conceivable with further optimizations to the feature set. Best viewed in color.

a small example in which we aim to detect a couch pillow in fifteen diverse scenes. Our method does not rely on any particular detection approach, but since we are interested in real-time detection, we feed our labeled pillow examples into the LINE-MOD template-matching framework described in Hinterstoisser et al [9]. At training time, LINE-MOD uses the depth and image cues to build templates (Figure 12, upper-left). At testing time, LINE-MOD searches for those templates in the previously unseen images and identifies any "detection" as a template match with a response exceeding some threshold (0.75 in our experiment). As can be seen in the illustration, we detect all objects while avoiding any false positives using a training set collected online with the STRGEN implementation.

V. Conclusions

We have presented a novel method of segmenting and tracking deformable objects in RGBD, making minimal assumptions about the input data. We have shown this method makes a significant quantitative improvement over the most similar state-of-the-art work in segmentation and tracking of non-rigid objects, that real-time performance is feasible, and that an online and interactive version of the algorithm is suitable for collecting training examples of objects in unstructured environments. A complete solution to this task would have far-reaching ramifications in robotics, computer vision, and graphics, opening the door for easier-to-train and more reliable object recognition, model capture, and tracking-based semi-supervised learning. While there remains more work to be done before a completely robust solution is available for unassisted segmentation and tracking, we believe this approach to be a promising direction for future research.

References

[1] C. Aeschliman, J. Park, and A. Kak. A probabilistic framework for joint segmentation and tracking. In *CVPR*, 2010.
[2] X. Bai, J. Wang, D. Simons, and G. Sapiro. Video snapcut: robust video object cutout using localized classifiers. In *SIGGRAPH*, 2009.
[3] C. Bibby and I. Reid. Robust real-time visual tracking using pixel-wise posteriors. In *ECCV*, 2008.
[4] C. Bibby and I. Reid. Real-time tracking of multiple occluding objects using level sets. In *CVPR*, 2010.

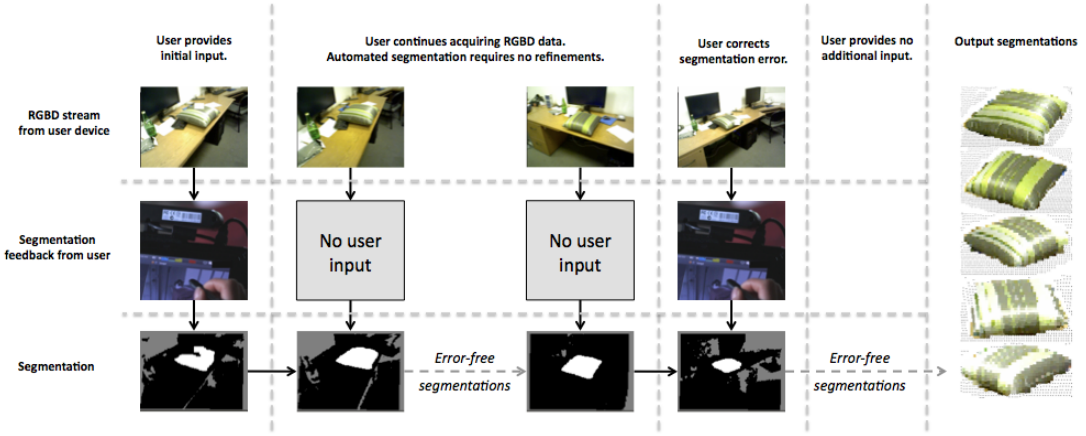

Fig. 11. Example data labeling pipeline demonstrating one possible application of the STRGEN algorithm. Segmented objects are obtained while the raw data is being recorded. Segmentation hints from the user generate node potentials in the CRF so the user can direct the segmentation at any time; without input, the segmenter makes use of only the other cues discussed in Section III-C to produce segmentation masks. The output is a set of segmented object examples suitable for training standard object detectors.

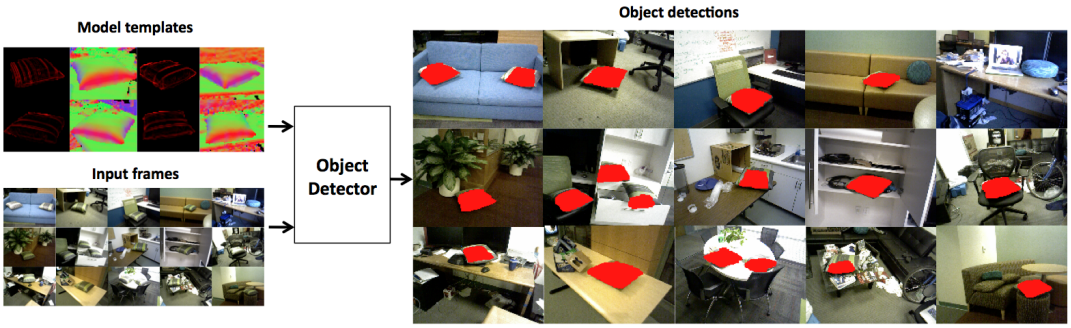

Fig. 12. Object detection example trained using the method in Figure 11. Color and depth templates are extracted from segmented training objects; an appropriate object detection method (LINE-MOD in our example) searches for the templates in previously-unseen test frames and identifies "detections" as matches with responses exceeding a threshold.

[5] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:359–374, 2001.

[6] I. Budvytis, V. Badrinarayanan, and R. Cipolla. Semi-supervised video segmentation using tree structured graphical models. In *CVPR*, 2011.

[7] M. Godec, P. Roth, and H. Bischof. Hough-based tracking of non-rigid objects. In *ICCV*, 2011.

[8] H. Grabner, M. Grabner, and H. Bischof. Real-Time Tracking via On-line Boosting. In *British Machine Vision Conference*, 2006.

[9] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *ICCV*, 2011.

[10] T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[11] Z. Kalal, J. Matas, and K. Mikolajczyk. P-n learning: Bootstrapping binary classifiers by structural constraints. In *CVPR*, 2010.

[12] J. Kwon and K. M. Lee. Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive basin hopping monte carlo sampling. In *CVPR*, 2009.

[13] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *CVPR*, 2007.

[14] A. Petrovskaya and S. Thrun. Model based vehicle detection and tracking for autonomous urban driving. In *Autonomous Robots*, 2009.

[15] V. A. Prisacariu and I. D. Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. In *BMVC*, 2009.

[16] X. Ren and J. Malik. Tracking as repeated figure/ground segmentation. In *CVPR*, 2007.

[17] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": interactive foreground extraction using iterated graph cuts. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 309–314, New York, NY, USA, 2004. ACM.

[18] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[19] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, 2011.

[20] S. Stalder, H. Grabner, and L. V. Gool. Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. In *International Conference on Computer Vision Workshop on On-line Learning for Computer Vision*, Sept. 2009.

[21] M. Szummer, P. Kohli, and D. Hoiem. Learning crfs using graph cuts. In *ECCV*, 2008.

[22] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *ICML*, 2005.

[23] A. Teichman, J. Levinson, and S. Thrun. Towards 3D object recognition via classification of arbitrary object tracks. In *International Conference on Robotics and Automation*, 2011.

[24] A. Teichman and S. Thrun. Tracking-based semi-supervised learning. In *Robotics: Science and Systems*, 2011.

[25] A. Teichman and S. Thrun. Learning to segment and track in rbgd. In *WAFR*, 2012.

[26] D. Tsai, M. Flagg, and J. Rehg. Motion coherent tracking with multi-label mrf optimization. In *BMVC*, 2010.

[27] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. 2005.

[28] C. Urmson. The Google Self-Driving Car Project, 2011.