Robust Tracking of Unknown Objects Through Adaptive Size Estimation and Appearance Learning

Alessandro Pieropan Niklas Bergström Masatoshi Ishikawa Danica Kragic Hedvig Kjellström

***Abstract*— This work employs an adaptive learning mechanism to perform tracking of an unknown object through RGBD cameras. We extend our previous framework to robustly track a wider range of arbitrarily shaped objects by adapting the model to the measured object size. The size is estimated as the object undergoes motion, which is done by fitting an inscribed cuboid to the measurements. The region spanned by this cuboid is used during tracking, to determine whether or not new measurements should be added to the object model.**

In our experiments we test our tracker with a set of objects of arbitrary shape and we show the benefit of the proposed model due to its ability to adapt to the object shape which leads to more robust tracking results.

I. INTRODUCTION

Many successful object tracking methods are typically based on a combination of detection, tracking and learning of the object model [1]. For applications in controlled environments, e.g. factory automation, a model of the object is built beforehand [2] reducing the problem to detection and tracking. However the process of building the model can be cumbersome. It requires either a specialized setting/equipment or a large amount of hand-labeled images. In both cases the amount of manual work required is high.

Alternatively, there are general methods that can track an object starting from a partial view and employing learning to build a complete representation of the object online [3], [4]. In such a case, the object is represented as a set of patches or interest points which describes the appearances of the object. New appearances are accumulated to the representation while the object moves or rotates.

Such an approach dramatically decreases the amount of supervision since the only required manual operation is the initialization, usually using a bounding box. However the method needs a mechanism to understand if an observed appearance belongs to the object. Incorporating the background as part of the object model increases the chance of failure of the tracker. This problem is often referred to as drifting due to the tracked region drifting over to the background.

In our previous work [5] we proposed a general tracking framework that can estimate the six degree of freedom (DOF) pose of an unknown object and simultaneously build a model of the object, represented as a sparse point cloud. An inscribed cuboid is used to bound the complexity of the learning procedure and mitigate the drifting problem occurring when learning the background. In addition to a manually provided initial bounding box, the tracker required knowledge of the depth of the object. Otherwise it was assumed that the depth was similar to the smaller of the initial (known) width and height.

In the present we move one step more towards generality by proposing a method that can adjust the learning mechanism dynamically to the estimated size of the object allowing our framework to work *without* knowing the size of the object beforehand.

The main contribution is an adaptive method that improves the stability of the tracker by allowing to learn the depth of the object online.

As the object begins to rotate, as in Fig. 1, we seek the depth of the object using the side corresponding to the face of the cuboid with greatest visibility. Then we employ a set of particles to explore the surface of the object and estimate

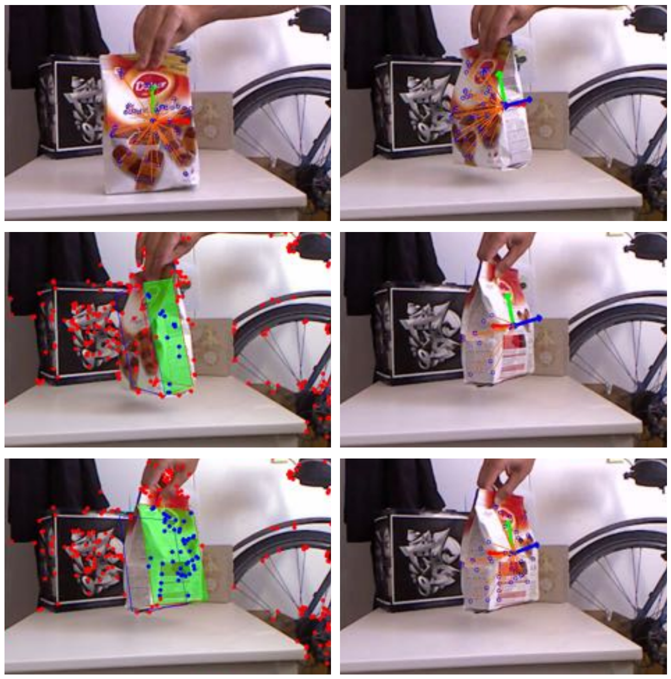


Fig. 1. Tracking and incremental learning. Initially (top left) the method knows only the appearance of the object facing the camera. Its depth is furthermore unknown. Current appearance measurements are used to estimate the pose of the object and together with depth measurements adjust the inscribed cuboid (right). As new sides of the object become visible (mid and bottom left), the cuboid is used to determine which measurements are added to the object model (blue points) and which are discarded (red points).

This research has been supported by the EU through ROBOHOW, IST-FP7-Collaborative Project-288533 and by the Japan Society for the Promotion of Science (JSPS).

AP, HK and DK are with CVAP/CAS, KTH Royal Institute of Technology, Stockholm, Sweden, pieropan,dani,hedvig@kth.se. NB and MI are with Ishikawa Watanabe Lab, University of Tokyo, Japan, niklas_bergstrom,ishikawa@ipc.i.u-tokyo.ac.jp.

The GPUs used for this research were donated by the NVIDIA Corporation.

the depth through consensus. The estimation is furthermore stabilized over time using a mean sliding window filtering technique. The result is then used to dynamically adjust the model of the object and learn new appearances accordingly.

Experiments show that the proposed method improves the robustness of the previously proposed tracking framework by introducing a more general approach through reducing the information needed in the initialization step.

The paper is structured as follows. After discussing the related work in Sec. II we present our adaptive tracker method in Section III. Quantitative experiments are presented in Sec. IV. Finally conclusions are discussed in Sec. V.

II. Related Work

Existing methods to track objects tend to be part of one of two categories. *Model-based methods* can robustly track an object whose appearance and shape are known beforehand. Some methods are able to estimate the pose of textureless objects relying solely on the shape or the edges of an object. This is particularly useful in an industrial environment to recognize metal parts. [6] proposes to track an object by rendering its model and compute the overlap with the edges found in the image. [7] is able to estimate the pose of an object with a monocular camera and its CAD model. [8] uses depth sensors to compute the pose directly in 3D space. Alternatively, in case the objects are textured, it is possible to compute the pose of an object by mapping the features of the object to the image [9]. However building the model of the object offline is a cumbersome procedure that requires a lot of manual work. It can be done using either a specific setting [10] or a large amount of hand tuned images [11].

On the other hand, there is a large spectra of *generic methods* able to track unknown objects. In [12] a method to track an object using mean shift is proposed, while a segmentation based method is used in [13]. A widely used approach is tracking-by-detection [14]–[16]. The localization of the object is performed using a bounding box as initialization and some kind of learning technique is employed to estimate the new position. In [17] boosting is employed while structured output SVM is used by [4]. The key aspect of all those methods is the ability to learn new appearances of the object allowing to update the model of the object online. However there is the risk that the learning strategy will include the background as part of the model causing the tracker to drift. Some methods are more prone to this since they apply an aggressive learning strategy [3]. Alternatively, no learning is used, and it is assumed that the appearance of the object does not change as in [18]. In this case however, the object is lost when this assumption does not hold.

All these methods require very little supervision but the object pose estimation is often expressed as an axis aligned bounding box. While this estimation can be enough for computer vision application (e.g. video surveillance), it is not enough for robotics application especially when the robot needs to interact with the object.

In our previous work [5], we presented a general tracker able to update the appearance of an object. Our learning strategy used an inscribed cuboid within the object to bound the complexity of the learning problem and prevent drifting. This is in the spirit of [19], where simple shapes such as cubes or cylinders are fitted within the object to learn new appearances and track it. Our method works robustly under the assumption that the depth of the object is known. [19] works if a simple shape can fit the actual object since the new appearances learned are projected on the surface of the shape. In the present work we want to move one step further having a completely adaptive approach that can work with objects that are very far from being rotationally symmetric.

III. Method

Our goal is to track an unknown object while it is moved, estimate its pose and accumulate new appearance measurements to update its representation. The object is expressed as a sparse cloud of 3D interest points $\mathbf{P}$ and its 6 DOF is described by its centroid $\boldsymbol{C}$ and the rotation matrix $\boldsymbol{R}$.

Given the initial position of the object in the first frame I_1 of sequence of RGBD images, the object pose is estimated through the rest of the sequence $I_2, ..., I_n$. Since the object shape is unknown and only one particular 3D view of the object is visible, the actual 3D size of the object has to be calculated during tracking. This is done by dynamically adjusting an inscribed cuboid representing the spatial extent of the object. Initially however, only the front of the object can be seen. Therefore tracking is initialized with a 2D bounding box representing the front face of the inscribed cuboid. The initial object model is represented as a sparse cloud of interest points $\boldsymbol{P}^j$ extracted within the area defined by the bounding box. Each 3D point $\boldsymbol{P}^j$ has a corresponding 2D keypoint $\boldsymbol{p}^j$ with an attached feature descriptor. The keypoints are used to calculate a sparse optical flow and perform feature matching. The first inscribed cuboid $\boldsymbol{IC}_1$ will be used in the learning mechanism.

The summary of our algorithm is shown Alg. 1, compared to our previous work [5], there are significant changes in the initialization and in the learning procedure. While we refer to the original paper for details on how optical flow and feature matching are employed to estimate the position and pose of the object, we here reiterate the visibility concept since it is central to the depth estimation as well as the learning of new appearances.

The following notation is used in the rest of the paper:

- A *frame* $I_i = (RGB_i, RGBD_i)$ consists of an RGB image and a the corresponding 3D points.
- $\boldsymbol{IC}$ correspond to the inscribed cuboid used for learning.
- $\boldsymbol{C}_i, \boldsymbol{R}_i$ is the 6 DOF pose of the object at time frame i.
- The *object model* consists of a sparse cloud of 3D points $\boldsymbol{P}$, 2D keypoints $\boldsymbol{p}$ and corresponding BRISK [20] feature descriptor.
- $\boldsymbol{V}_i^c, c \in \{F, L, R, Ba, T, Bo\}$ indicates the visibility of the faces of the cuboid (F=front, Ba=back, L=left, R=right, T=top, Bo=bottom) that are visible at frame i.
- An *observation* $\boldsymbol{\pi}_i = [\boldsymbol{\pi}_i^j] = ([\boldsymbol{p}_i^j], [\boldsymbol{P}_i^j])$ is an array of

```
Input: $I_1, ..., I_n, b_1$
Result: $b_2, ..., b_n$, $IC_2, ..., IC_n$
$\boldsymbol{IC}_1 \leftarrow$ init_cube($b_1$);
$\boldsymbol{\pi}_1 \leftarrow$ extract_points($I_1$);
$\boldsymbol{\Pi} \leftarrow$ label_points($\boldsymbol{\pi}_1, b_1$);
$c \leftarrow F$;
for $i \in 2:n$ do
    $\boldsymbol{\pi}_{i-1} \leftarrow \boldsymbol{\Pi}^V_{i-1}$;
    $\tilde{\boldsymbol{\pi}}_i \leftarrow$ track_points($I_i, I_{i-1}, \boldsymbol{\pi}_{i-1}$);
    $\boldsymbol{R}_i \leftarrow$ get_rotation($\tilde{\boldsymbol{\pi}}_i, \boldsymbol{\Pi}$);
    $\boldsymbol{v}_c \leftarrow$ vote($\tilde{\boldsymbol{\pi}}_i, \boldsymbol{R}_i, \boldsymbol{\Pi}$);
    $\boldsymbol{C}_i \leftarrow$ dbscan($\boldsymbol{v}_c, \varepsilon, \lambda$);
    $\{\boldsymbol{IC}_i, \boldsymbol{b}_i\} \leftarrow$ update_cuboid($\boldsymbol{IC}_{i-1}, \boldsymbol{C}_i, \boldsymbol{R}$);
    $\hat{\boldsymbol{\pi}}_i \leftarrow$ extract_points($I_i$);
    $\hat{\boldsymbol{\pi}}^m_i \leftarrow$ match_points($\boldsymbol{\Pi}^{V^c}, \hat{\boldsymbol{\pi}}_i$);
    $\boldsymbol{\pi}_i \leftarrow$ update_point_set($\tilde{\boldsymbol{\pi}}_i, \hat{\boldsymbol{\pi}}^m_i$);
    $\tilde{\boldsymbol{V}} \leftarrow$ get_visibility($\boldsymbol{R}$, $\boldsymbol{C}$);
    $\delta \leftarrow$ estimate_size($\boldsymbol{IC}_i$);
    $\boldsymbol{IC}_i \leftarrow$ adjust_cuboid($\boldsymbol{IC}_i, \delta$);
    $\boldsymbol{\Pi} \leftarrow$ learn($\boldsymbol{\Pi}, \hat{\boldsymbol{\pi}}_i \backslash \hat{\boldsymbol{\pi}}^m_i, \tilde{\boldsymbol{V}}, \boldsymbol{R}_i, \boldsymbol{IC}_i$);
    $c \leftarrow$ pick_visible($\tilde{\boldsymbol{V}}$);
end
```
Algorithm 1: Pipeline of the algorithm.

keypoints with attached feature descriptors $\boldsymbol{p}_i$ and corresponding 3D points $\boldsymbol{P}_i$ extracted at frame i.
- $\boldsymbol{\Pi}^{V^c}$ corresponds to the subset of points that has been extracted using the faces $\boldsymbol{V}^c$.
- $\boldsymbol{D}_i = \boldsymbol{d}^j_i$ is the set of particles used to estimate the size δ of the object.
- $\boldsymbol{\mu}^i$ is the normal vector of face i.
- $\boldsymbol{F}^i$ is the set of appearance measurements extracted from an image frame delimited by a face of the inscribed cuboid.
- Subscript i refers to the number of the image in the sequence, superscript j refers to the index of a point. If only the subscript is used the variable indicates an array.

A. Initialization

The manually provided bounding box $\boldsymbol{b}_1$ is used to initialize the object model. BRISK features [20] are detected in the initial frame and the corresponding keypoints are extracted $\boldsymbol{p}_i$. Each keypoint is associated to its corresponding 3D point $\boldsymbol{P}_i$. All the points p_i, P_i within the space defined by $\boldsymbol{b}_1$ are added to the model of the object only if $\boldsymbol{P}^j_i$ has a valid depth value, and discarded otherwise. Points can have invalid depth e.g. if they are located close to the object boundary or in very reflective areas. All the keypoints outside the area delimited by the bounding box are marked as *background*. The initial bounding box $\boldsymbol{b}_1$ is also used to initialize the front face of the inscribed cuboid and to position it within the object. The size of this cuboid will subsequently be updated as the object starts to move. Its sole purpose is to prevent including background into the object model by only learning new appearances that reside *within* the cuboid.

B. Visibility of the cuboid

Depth estimation and learning new appearances depends largely on to what extent a certain face c of $\boldsymbol{IC}$ is facing the camera. We refer to this as the *visibility* V^c of the face. The pose of the inscribed cuboid is given by the estimated rotation matrix of the object $\boldsymbol{R}_i$ and the computed object centroid $\boldsymbol{C}_i$. It is then possible, using the normal of the faces $\boldsymbol{V}$, to calculate the area of each face visible to the camera as in Eq. (1). A negative value means that the object is facing away from the camera.

$$V^c = \boldsymbol{\mu}^c * \boldsymbol{R}_i[:,2] \in [-1,1] \tag{1}$$

The visibility concept is illustrated in Fig. 2. The tracker is initialized in the first image, so only the front face of the cuboid is visible to the camera (blue). As soon as the object rotates, the cuboid is rotated according to $\boldsymbol{R}_i$ and it is translated according to the newly calculated $\boldsymbol{C}_i$. The right face of the cuboid (green) becomes more visible.

C. Evaluating the current pose

As discussed in the introduction, a tracker that learns the appearance of the object is prone to drifting caused by the inclusion of the background in the object model. A very aggressive learning technique tends to drift more than a conservative one. Therefore it is crucial to define a good mechanism to learn appearance candidates. The use of the inscribed cuboid allows our method to filter out the background. However this alone is not enough since the estimated object pose may be incorrect. As a result the cuboid may be not inscribed in the object and may include parts of the background. To detect such scenarios we analyze the temporal signature of the estimated pose in order to detect sudden changes that are likely to correspond to errors. In order to do that we keep a history of the estimation in the past t frames represented as a set of the quaternions $\boldsymbol{q}_i$

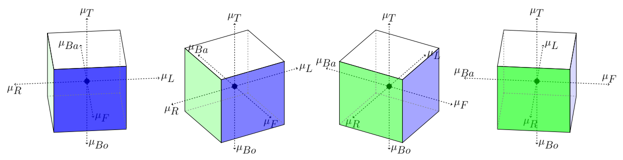


Fig. 2. In the first image the front face is facing the camera, while the right face is barely visible. As the object rotates the right face becomes more visible while the visibility of the front face decreases.

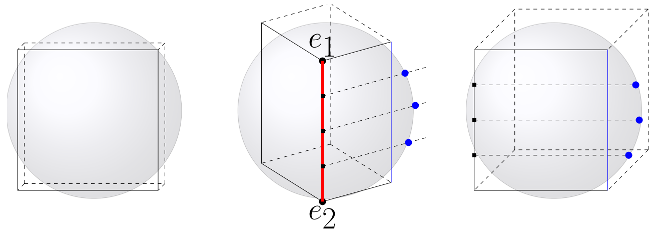


Fig. 3. Depth estimation. Particles are spawned on the surface of the object starting from the visible edge of the cuboid (red edge). They will explore the surface following a linear path. Each particle will then vote for the estimated depth and the cuboid will be adjusted accordingly (blue edge).

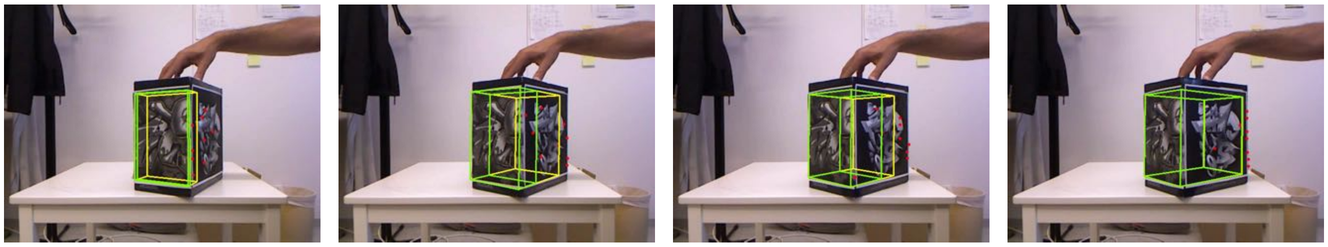


Fig. 4. Example showing the estimation of the depth using particles. In the initial frame the object is slightly rotated and the particles starts to explore the object. As soon as the object rotates the inscribed cuboid starts to increase. Due to noise in the depth data some particles do not provide a provide a good estimate. Therefore the average (yellow box) is unstable. The sliding window estimation (green box) is more consistent and stable over time.

extracted from the estimated $\boldsymbol{R}_i$. We compute the median angular distance with the quaternions computed in the last t frames. Therefore the computation of a low median angular distance ΔQ indicates a robust estimated pose while a high value represent a noisy estimation. Our learning mechanism and depth estimation are triggered only if the value of this signature is lower than a certain threshold γ. More details can be found in our previous work [5].

D. Estimating the depth of the object

After initialization, the front face F of the cuboid is visible as shown in Fig. 2 and Fig. 3. The depth of the object is still unknown. Assuming orthographic projection, the depth can be completely known as soon as the objects yaws or pitches by 90 degrees, corresponding to a complete side view with respect to the initial position. We use this fact to continuously estimate the depth of the object as it starts rotate until a 90 degree rotation is reached. Upon rotation, a set of particles $\boldsymbol{D}$ is initialized on the surface of the object with the purpose of estimating the depth. We found that a particle based approach, described below, is more robust to noise in the estimation if compared to a segmentation approach such as *connected components* [21]. As we have shown in [22], a segmentation approach is very sensitive to the proximity of other objects. A segment can propagate to objects interacting with the tracking target (e.g. human hand rotating the object) causing to overestimate the size of the object. With a particle based approach, there is indeed the possibility that some particles will spread over another entity, but the voting mechanism will filter out these.

Each particle $\boldsymbol{p}^k$ independently estimates the object depth and votes for δ_i, the depth of the object in frame i. As explained in Sec. III-C, such estimation is performed only when the estimated pose of the object in the current frame is considered to be reliable.

First the visibility of each face $V_i^c, c \in \{R,T,Bo,L\}$ is calculated as in Eq. (1), where a value of one represent a face straight in front of the camera. Particles are then initialized on the edge $\boldsymbol{e} = [e_1, e_2]$ shared between face F and the face with the highest visibility, which is chosen as follows:

$$c = \underset{c}{\operatorname{argmax}} \boldsymbol{V}^c \text{ with } c \in \{R,T,Bo,L\}. \quad (2)$$

Fig. 3 illustrates this procedure. The initial position of the n particles on the edge $\boldsymbol{e}$ in image coordinates is defined by the following equation:

$$\begin{aligned} \boldsymbol{d}_1 = \{d_1^k[x,y] \mid x = x_{e_1} + k * \frac{x_{e_2} - x_{e_1}}{n}, \\ y = y_{e_1} + k * \frac{y_{e_2} - y_{e_1}}{n} \\ (\text{where } 0 < k < n)\} \end{aligned} \quad (3)$$

For a frame at time i The particles are then moved independently in the image in the direction of a vector $\vec{\boldsymbol{v}}$, perpendicular to the edge e until the stop criteria has been reached.

$$d_{s+1}^k = d_s^k + \alpha\vec{v} \quad (4)$$

The stop criteria is defined as follows. The 3D points corresponding to the particles $\boldsymbol{d}_s^k$ are projected to points $\boldsymbol{D}_s^k$ on the surface of the object along the normal of the selected face c. If there is a discontinuity along the resulting 3D path $\boldsymbol{d}_1^k,\ d_2^k, ..., d_s^k$, the last particle is taken as one vote for the estimated depth, and the particle update stops. The path is considered to be discontinuous at step s if

$$\|\boldsymbol{D}_s^k - \boldsymbol{D}_{s+1}^k\| > \iota \quad (5)$$

The estimated depth δ_i at frame i is the median of the measurements obtained by the n particles. Fig. 3 shows an example where 3 particles are spawned along $\boldsymbol{e}$. When the particles meet a discontinuity they stop and the last valid depth reached is used to compute the median depth of the object for that specific time frame.

Since the variance of the depth estimation is high due to noise in the observations, we employ a temporal sliding window mean filter technique,

$$\hat{\delta}_i = \frac{1}{t} \sum_{k=i-t+1}^{i} \delta_k, \quad (6)$$

to make the estimation stable over time. Fig. 6 shows the estimation of the depth of an object while it is rotating. It can be noticed that the estimation performed at a specific time point (yellow line) is noisy before the object reaches a considerable side view (red line) but the learning mechanism already accumulates new appearances used to track the object while the estimation is noisy. The sliding window average smooths the signal (green line) providing a more stable depth estimation to the learning procedure.

Fig. 4 shows an example of the estimation. The yellow box

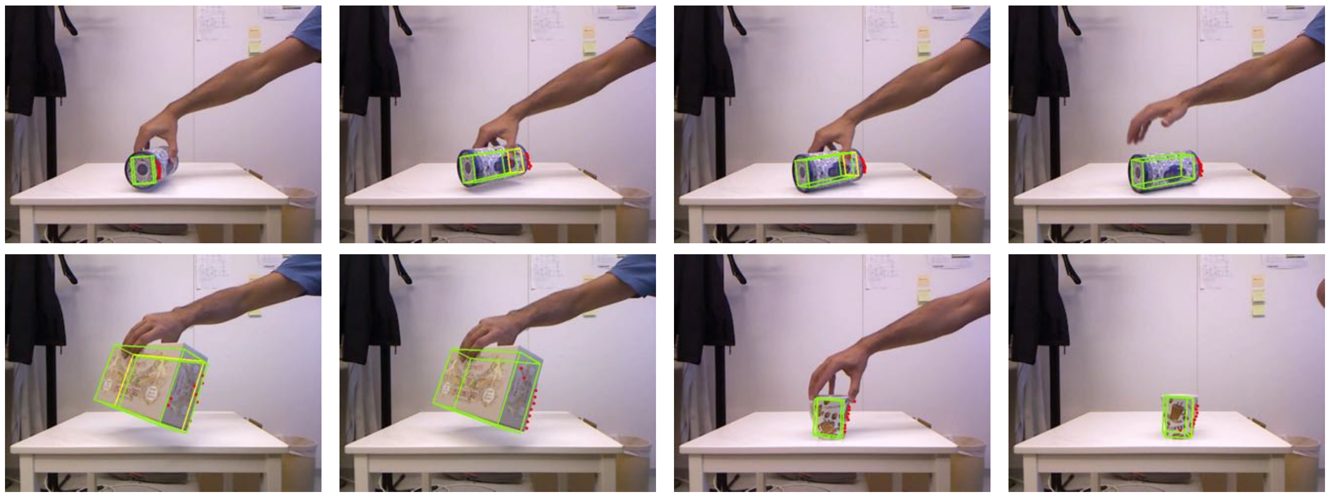


Fig. 5. Results achieved using the proposed method to estimate the size of the object online. The yellow estimation in the result obtained trough voting in the current frame while the more stable green line shows the sliding window average over time. The single frame estimation stabilizes the closer object is to a full size view as explained in Fig.6.

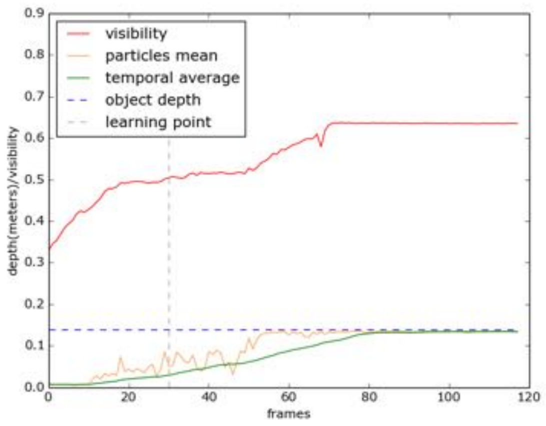


Fig. 6. Plot showing an example of the estimation. The object is rotated so that the visibility of the newly visible face increases (red line). The estimation performed with the particles (yellow line) is very unstable until a certain visibility is reached due to noise and reflectance. The temporal average (green line) is smoother and it converges to the actual depth (blue line). The gray dashed line indicates the time frame when the learning procedure starts since the minimum visibility required is met.

shows the depth estimation in the current frame, while the green box shows the estimation performed with the sliding window average. It can be noticed that the average over time $\hat{\delta}_i$ increases steadily and stabilize around the actual depth of the object.

E. Learning new appearances and updating the model

An appearance candidate that can be included in the model is not evaluated only in terms of the stability of the estimation (Sec. III-C). The model is updated only if the quality of the new candidate F^i_{new} is higher that the current learned face F^i_{old}. The quality is determined by the current visibility value V^i. The closer the value is to 1, the closer the object is to the ideal situation of having a completely new part of the object in front of the camera meaning that we are confident that an accurate depth is estimated.

$$F^i = \begin{cases} F^i_{new}\; if\; V^i_{new} > V^i_{old} \wedge \Delta Q^i_{new} < \Delta Q^i_{old} \\ F^i_{old}\; otherwise \end{cases} \tag{7}$$

Fig. 7 shows an example of the procedure in case the quality of the current model is higher to what has been learned so far. First the inscribed cuboid is adjusted to the newly estimated size (blue box). Secondly, 3D points $\mathbf{\Pi}_i$, keypoints $\boldsymbol{\pi}_i$ and feature descriptors are extracted from the area of the object contained in the updated inscribed cuboid. Then they are incorporated in our model using the following equation:

$$\boldsymbol{rm}^j_1 = \boldsymbol{R}_i^{-1}(\boldsymbol{P}^j_i - \boldsymbol{C}_i) \tag{8}$$

where $\boldsymbol{R}_i, \boldsymbol{C}_i$ is the estimated pose of the object in the current frame. $\boldsymbol{rm}$ is the position of the point of interest extracted with respect to the pose of the object in the initial frame $\boldsymbol{R}_1, \boldsymbol{C}_1$. Note here the function of the inscribed cuboid which decides which 3D points from $RGBD_i$ that are included in the model update.

In this section we described the adaptive learning method that we employ in our tracking framework. We now move on to the experiment section where we test our tracker on a set

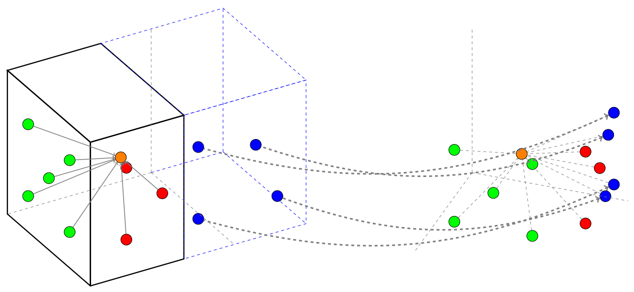


Fig. 7. Points in green and red are currently used to track the object and calculate its pose. They are in different color because they are associated to different faces of the cuboid. During the learning procedure the tracker estimated an increased size of the object and the cuboid is updated (in blue). New keypoints are extracted from the updated part (blue points) and added to the model.

of objects of arbitrary shape. As an example Fig. 1 shows the tracker working with a bag of cookies while in Fig. 5 a cup is being tracked. The pose estimated by our framework will be evaluated against our previous method.

IV. Experiments

We perform qualitative experiments to test the robustness of the proposed method compared to our previous work [5]. First we describe how the parameters introduced are set in Sec. IV-A, then we evaluate our method in terms of robustness of the pose estimated while the object moves in Sec. IV-B. Finally we analyze the framework computing time and its scalability to potentially track multiple objects.

A. Parameter settings

The proposed method starts to learn new appearances when an unknown face is visible more that a value $\kappa > 0.5$. The quality measurement γ of a learning candidate based on the median quaternion distance has to be lower than 10 degrees. The number of particles n generated to estimate the depth are 10. Each particle stops exploring the surface of an object if the distance between two consecutive 3D points ι is less than 1 cm.

B. Benchmark

The goal is to evaluate the behavior of the tracker when the rotational symmetric assumption does not hold. There are two different scenarios where this criteria is not met: The object depth is (1) smaller or (2) bigger than the observed width or height that are used to guess the depth in the previous method. In the first row of Fig. 10(a,b,c) we show some results achieved using objects that fall in category (1). The trackers are initialized with a bounding box within the boundary of the visible area of the object. When the object yaws, the learning procedure starts to accumulate new appearances of the objects. The proposed method adapts the dimension of the inscribed cuboid according to the estimated depth. It can be seen that the cuboid grows until it converges to the real size of the object. Features are extracted from the area of the object delimited by the lateral side of the cuboid, increasing the number of features on that side used for tracking. In the second row of Fig. 10(a,b,c) the previously proposed tracker is used. The cuboid in those examples is not adjusted and some parts of the background are inside the area delimited by the cuboid. Therefore appearances of the background are included in the model of the object upon learning, causing the tracker to drift. We use objects that fall in category (2) in Fig. 11(a,b). When the object yaws the tracker increases the size of the cuboid allowing to extract features from a bigger portion of the object making the result more stable. In the second row of Fig. 11(a,b) our previous method is used. It can be noticed in Fig. 11(b) that after a 270 degrees rotation the previous tracker estimates the pose incorrectly since not enough appearance measurements are learned for tracking.

Fig. 8 shows a plot comparing the number of features used for tracking in the experiment in Fig. 10(c). This visualizes the importance of having a correct estimate of the object size in order to maintain good tracking.

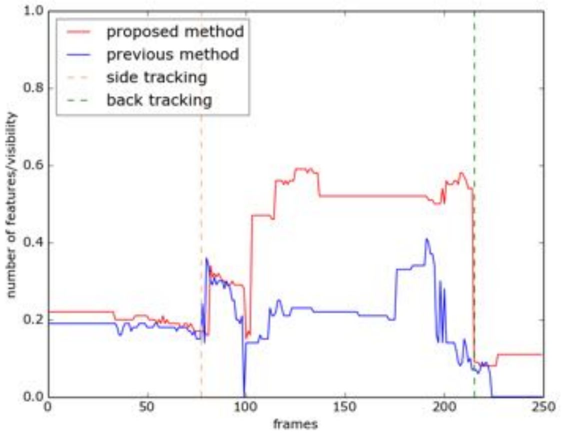
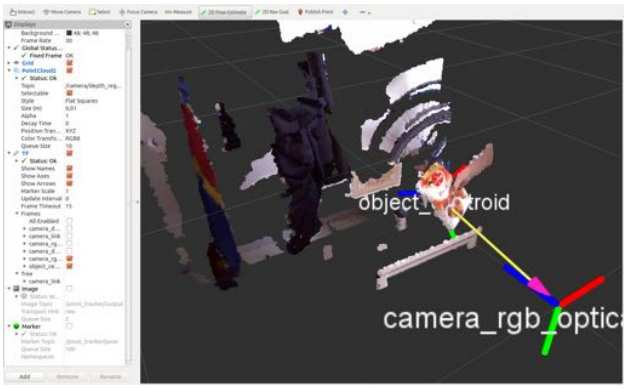


Fig. 8. Comparison of the proposed method with our previous tracker using an elongated object as in Fig. 10(c). As soon as the object rotates on the side more features are learned. The yellow dashed line indicates the time point when then tracker starts to use the features on the side. Our method can use more features and have a more robust pose estimation. Upon tracking the back face of the object, indicated by the green dashed line, our previous method loses the object since there are not enough features to estimate the pose.



```
Background ...    48; 48; 48
Frame Rate        30
Global Status...
  Fixed Frame     OK
Grid
PointCloud2
  Status: Ok
  Topic           /camera/depth_reg...
  Selectable
  Style           Flat Squares
  Size (m)        0.01
  Alpha           1
  Decay Time      0
  Position Tran...  XYZ
  Color Transfo...  RGB8
  Queue Size      10
TF
  Status: Ok
  Show Names
  Show Axes
  Show Arrows
  Marker Scale    1
  Update Interval 0
  Frame Timeout   15
  Frames
    All Enabled
    camera_d...
    camera_link
    camera_rg...
    camera_d...
    camera_rg...
    object_ce...
  Tree
    camera_link
Image
  Status: W...
  Image Topic     /pinot_tracker/output
  Transport Hint  raw
  Queue Size      2
Marker
  Status: Ok
  Marker Topic    /pinot_tracker/pose
  Queue Size      100
  Namespaces
```

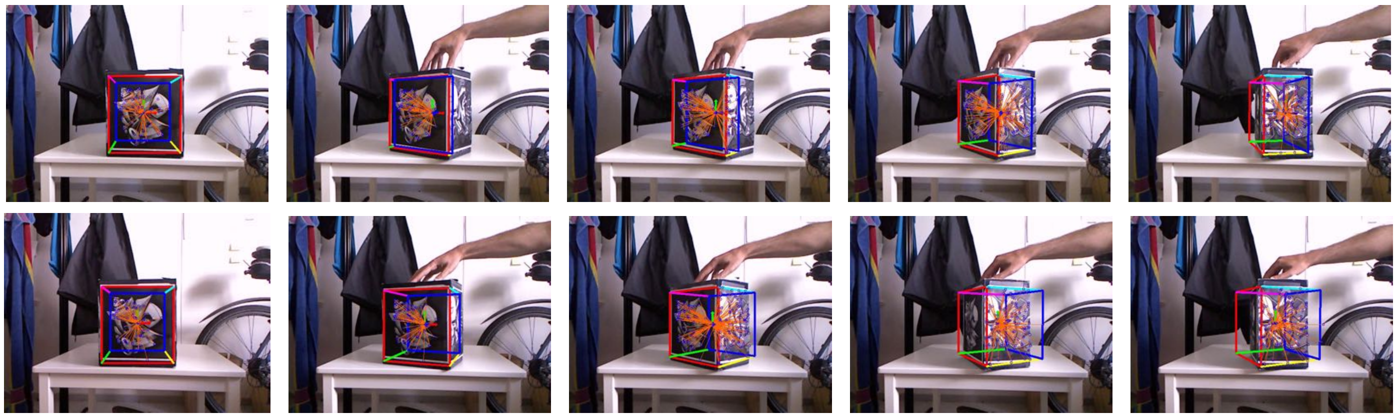
Fig. 9. ROS tracking framework preview.

C. Performance

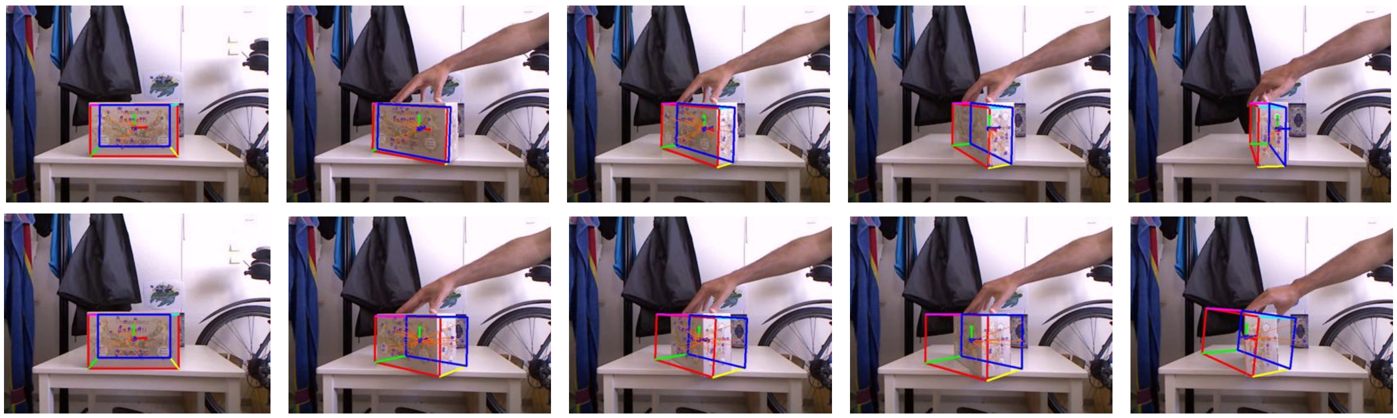
The current implementation of the algorithm has been tested on a desktop computer with an Intel core $i73687U$ using a single core. The average running time to estimate the pose of an object with an RGB-D frame extracted with Kinect 1 is 13 milliseconds. The average running time with Kinect 2 is 28 milliseconds.

D. Scalability

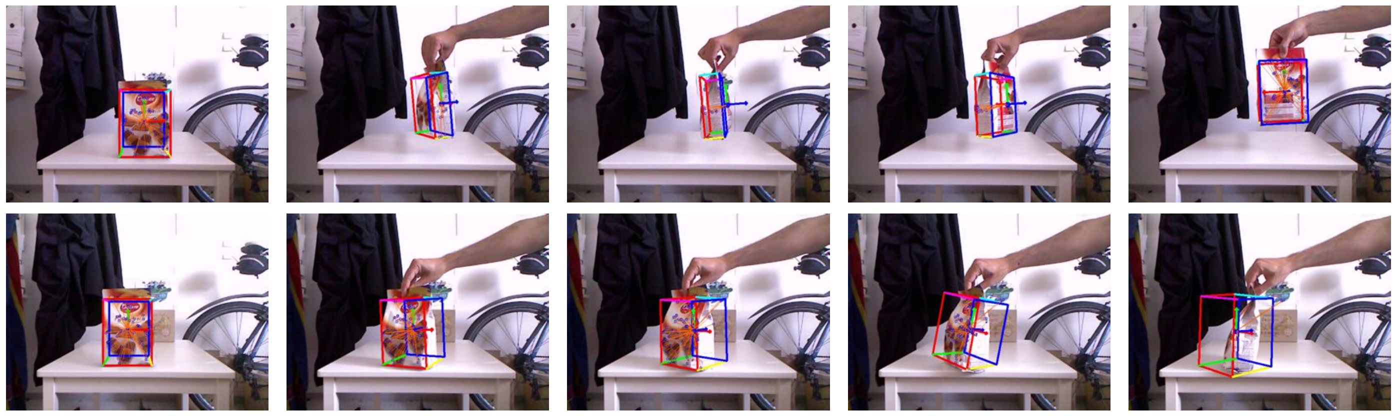
The most time consuming operations performed by the tracker are the computation of sparse optical flow and the extraction of features in an image. Those operation are performed globally on the image, so the performance of the tracker will not be greatly affected by an increasing number of objects. The important factor that influence the performances are the number of features extracted from an



(a) Wide box: the object depth is shorter than its width and height. The results of the proposed method are shown in the first row.



(b) Wide box: the object depth is shorter than its width and height. The results of the proposed method are shown in the first row.



(c) Pack of biscuits: arbitrary shaped object with its depth smaller than its width or depth. The results of the proposed method are shown in the first row.

Fig. 10. Results achieved using the proposed method against our previously presented tracker [5] using objects that fall in category (1). The depth of the object to track is smaller than its width or height therefore our original method started to add appearance measurements of the background to the object model causing the tracker to drift.

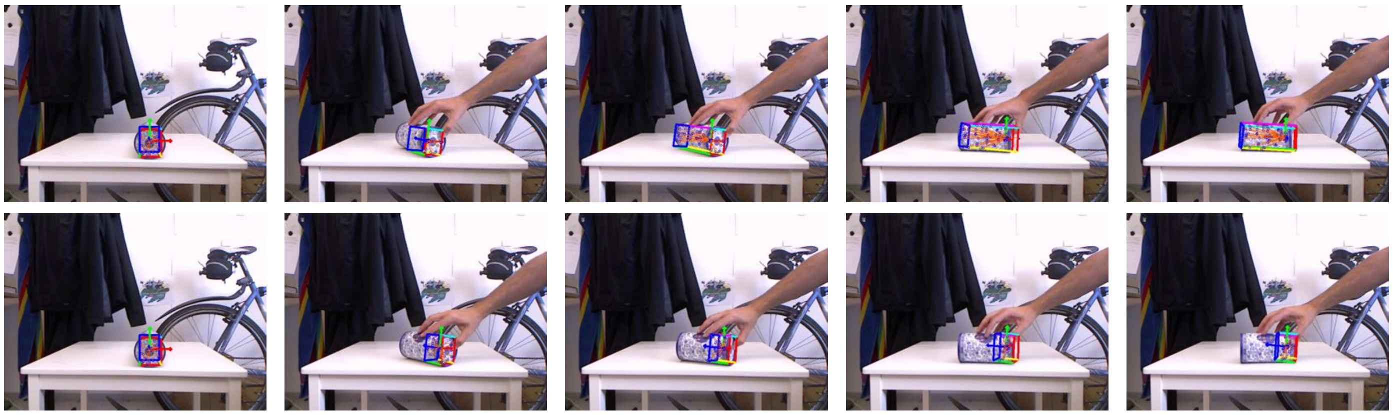
image. Highly textured objects and very cluttered environments allow to extract more features increasing the cost of matching the features and clustering the votes. However both steps could easily be parallelized on a graphics card.

E. Tracking Software Package

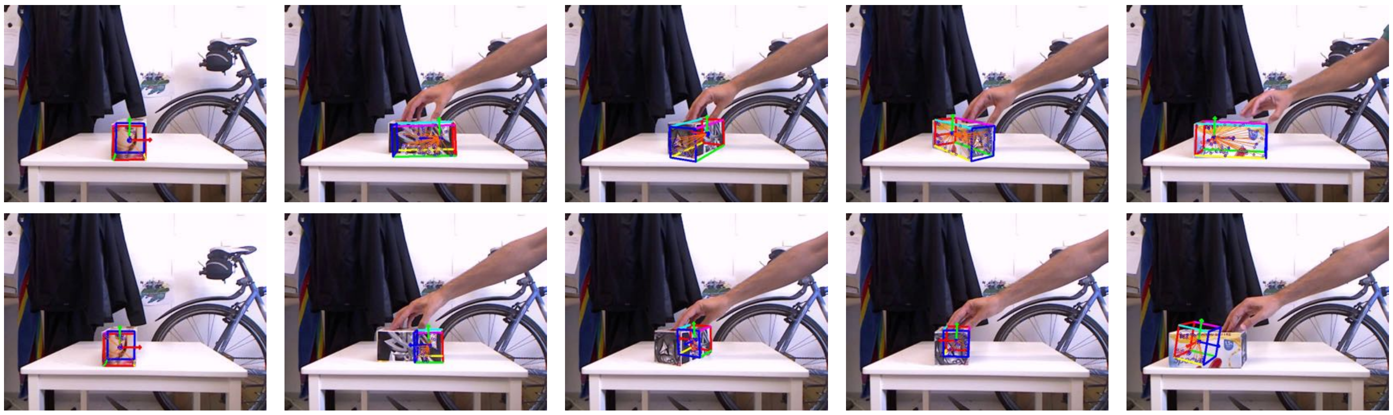
The proposed method is part of a larger ROS-based open source package that will be released under the BSD-license. The package allows to perform 2D tracking of unknown object with monocular cameras [23] and 3D tracking with RGBD camera. We also provide code to use different cameras with the tracker and detailed instructions how to use it.

V. Conclusions

In this paper we present a framework to detect, learn and track unknown 3D objects that estimates the size of the object online and adapts its learning strategy accordingly. Our method is able to track online an unknown object of an arbitrary shape with an average running time of 13 mil-



(a) Elongated cylinder: rounded shape object with the depth bigger than its height or width. The results of the proposed method are shown in the first row.



(b) Elongated box: the object is rotated 270 degrees on the y axis. The results of the proposed method are shown in the first row.

Fig. 11. Results achieved using objects that fall in category (2). The object depth is bigger than expected therefore the tracker has a more robust estimate of the pose adjusting the cuboid and extracting more feature points.

liseconds using Kinect 1. We compared our method against our previous work [5] showing consistent improvements in the estimation of the pose making the tracker less prone to drift. Our method is part of a bigger ROS-based package that will be released soon allowing researchers to quickly track unknown objects in experiments without the burden of building the model of the object.

References

[1] Y. Wu, J. Lim, and M-H. Yang. Online object tracking: A benchmark. *CVPR*, 2013.

[2] A. Pieropan, G. Salvi, K. Pauwels, and H. Kjellström. Audio-visual classification and detection of human manipulation actions. In *IROS*, 2014.

[3] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *PAMI*, 7(34):1409–1422, 2012.

[4] S. Hare, A. Saffari, and P. H. S. Torr. Struck: Structured output tracking with kernels. In *ICCV*, 2011.

[5] A. Pieropan, N. Bergström, M. Ishikawa, and H. Kjellström. Robust 3d tracking of unknown objects. In *ICRA*, 2015.

[6] P. Azad, D. Munch, T. Asfour, and R. Dillmann. 6-dof model-based tracking of arbitrarily shaped 3d objects. In *ICRA*, 2011.

[7] M. Ulrich, C. Wiedemann, and C. Steger. Cad-based recognition of 3d objects in monocular images. In *ICRA*, 2009.

[8] T. Schmidt, K. Hertkorn, R. Newcombe, Z. Marton, M. Suppa, and D. Fox. Depth-based tracking with physical constraints for robot manipulation. In *ICRA*, 2015.

[9] Karl Pauwels and Danica Kragic. Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking. In *IROS*, 2015.

[10] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. Bigbird: A large-scale 3d database of object instances. In *ICRA*, pages 509–516, May 2014.

[11] Autodesk. 123d catch. `http://www.123dapp.com/catch`, 2015.

[12] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000.

[13] N. Bergström, M. Björkman, and D. Kragic. Generating object hypotheses in natural scenes through human-robot interaction. In *IROS*, 2011.

[14] S. Avidan. Support vector tracking. *PAMI*, 26(8):1064–1072, 2004.

[15] S. Avidan. Ensemble tracking. *PAMI*, 29(2):261–271, 2007.

[16] R.T. Collins, Yanxi L., and M. Leordeanu. Online selection of discriminative tracking features. *PAMI*, 27(10):1631–1643, 2005.

[17] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *BMVC*, 2006.

[18] G. Nebehay and R. Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *WCACV*, 2014.

[19] T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, and M. Vincze. Blort-the blocks world robotic vision toolbox. *Proc. ICRA Workshop Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010.

[20] S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints. *ICCV*, November 2011.

[21] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *ACM*, 1992.

[22] A. Pieropan and H. Kjellström. Unsupervised object exploration using context. In *ROMAN*, 2014.

[23] A.Pieropan, N.Bergström, H.Kjellström, and M.Ishikawa. Robust and adaptive keypoint based object tracking. *Advanced Robotics*, 2015. In press.