Robust 3D Tracking of Unknown Objects

Alessandro Pieropan Niklas Bergström Masatoshi Ishikawa Hedvig Kjellström

***Abstract*— Visual tracking of unknown objects is an essential task in robotic perception, of importance to a wide range of applications. In the general scenario, the robot has no full 3D model of the object beforehand, just the partial view of the object visible in the first video frame. A tracker with this information only will inevitably lose track of the object after occlusions or large out-of-plane rotations. The way to overcome this is to incrementally learn the appearances of new views of the object. However, this bootstrapping approach is sensitive to drifting due to occasional inclusion of the background into the model.**

In this paper we propose a method that exploits 3D point coherence between views to overcome the risk of learning the background, by only learning the appearances at the faces of an *inscribed cuboid*. This is closely related to the popular idea of 2D object tracking using bounding boxes, with the additional benefit of recovering the full 3D pose of the object as well as learning its full appearance from all viewpoints.

We show quantitatively that the use of an inscribed cuboid to guide the learning leads to significantly more robust tracking than with other state-of-the-art methods. We show that our tracker is able to cope with 360 degree out-of-plane rotation, large occlusion and fast motion.

I. Introduction

Tracking of unknown objects is a heavily researched topic within the robotics and computer vision communities. This indicates the importance of the task, which is an essential prerequisite for applications like activity recognition, human-robot interaction and robot target pursuit. Still, little research on e.g. activity recognition actually employs fully working general tracking systems. Instead, scenes with simplified environments [1] or trackers for specific objects [2] are used to transform videos into a high-level representation suitable for the task at hand. Nonetheless, for realistic activity recognition in a general and unstructured environment to be possible, robust and versatile visual tracking of objects is of great importance. To this end, we present a visual tracker that is able to learn the appearance of unknown objects and robustly track not only their position, but their full 3D pose.

For specific applications in controlled environments, e.g. factory automation, the tracking is aided by the fact that 3D models of the objects to be tracked can be constructed beforehand. However, in a general scenario, the tracker does not have a full 3D model of the object initially, just a partial model that can be created from the first video frame. One approach is to settle with the initial model and track the object based on this [3]. This is a quite robust approach, given existing robust methods for object category detection and image segmentation. However, the downside is naturally that unknown faces of the object will remain unknown, which means that the tracker will lose track of the object in the case of a large orientation change.

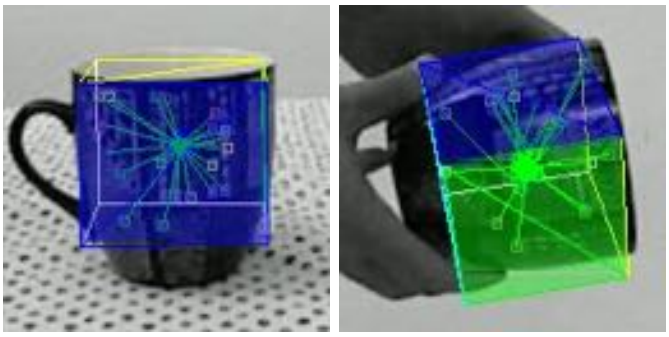


Fig. 1. Tracking and incremental learning. Initially the method knows only the appearance of the object facing the camera but the appearance of other views are estimated during tracking of the moving object.

The way to overcome this problem is to continuously learn new faces of the object as it undergoes transformations in the scene [4], [5]. While this intuitively seems like the natural approach, an aggravating factor is that the method itself needs to determine which parts of the newly observed appearances belong to the object and which belong to the background. Failure in this assessment might lead to the background being interpreted as part of the object and wrongly incorporated in the 3D model. Once this happens the risk of tracking failure increases and tracker has higher difficulty recovering when it loses track.

In recent years, several methods using a tracking-by-detection paradigm has been proposed, including methods combining this paradigm with regular tracking using optical flow [4]. These methods define the problem as finding the position of a 2D bounding box that best fit around the object given some criteria over the learned model. Depending on the method, the bounding box is fixed in size and orientation [4], [5], or adapts to transformations of the object [3].

With the introduction of affordable 3D image sensors a natural extension of the 2D bounding box trackers is a tracker based on 3D *inscribed cuboids*[1]. In this paper we present an adaptive tracker that extends the 2D paradigm by *estimating the rotation, size and position of the bounding box to include the full 3D pose of the tracked object*. At its core is a

This research has been supported by the EU through TOMSY, IST-FP7-Collaborative Project-270436, the Swedish Research Council (VR), and by the Japan Society for the Promotion of Science (JSPS).

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

[1]We use *inscribed* rather than *bounding* since we do not require the cuboid to fully encompass the object.

keypoint-based tracker employing a combination of optical flow and tracking-by-detection. While tracking occurs in 2D, each keypoint has an assigned depth value from the depth sensor, so the 3D position of each keypoint is known. Furthermore, the relation of the object center to each keypoint is known, making it possible to track the object with respect to both position, orientation, enabling continuous recovery of the full 3D object pose. Moreover the defined cuboid bounds the complexity of the learning procedure since the appearances required for tracking are limited by the number of faces. This centroid is defined initially through a manually provided inscribed cuboid, analogous to the bounding box, in the first frame. We show through experiments the benefits of the method, including seamless tracking of 360 degree out-of-plane rotations.

The main contributions are:

- A structured approach to learn previously unseen faces of a tracked object.
- A tracker that enables full 3D pose recovery.
- A natural bound on the complexity of the tracker.

The paper is structured as follows. In Sec. II we give an overview of related work. In Section III the method is described in detail. In Sec. IV we present a quantitative evaluation of the proposed framework. Finally conclusions and potential future directions are discussed in Sec. V.

II. Related Work

In scenarios where it is known beforehand that only a limited set of objects will be tracked, incorporating prior knowledge of the object in the method can help solving the problem [6]. In the general case however, a robot must be prepared to deal with previously unseen objects.

There exist a large spectra of approaches to tracking unknown objects, e.g. target boundary tracking [7], using mean shift [8] or using a segmentation based approach [9]. The predominant methodology, which also the proposed method adopts, is locating the object using a bounding box. For an extensive overview and benchmark of such methods we direct the reader to [10]. Below we discuss a few of them along with a few newer methods and how they relate to the proposed method.

The proposed tracker uses 3D data and estimates the position and pose of a 3D cuboid. In [11] the authors set out to do the same thing. However, their tracker performs the tracking directly in the 3D point cloud. In this sense the proposed tracker is more similar to the bounding box-based approaches since the *tracking of features* is done in 2D while the resulting *tracking of the object* is done in 3D.

As mentioned in Sec. I, one common approach is to adopt the tracking-by-detection methodology [12]–[14]. These methods often employ some learning framework such as boosting [15] or structured output SVM [5]. A crucial aspect of such methods is the ability to learn new appearances of the tracked model online. However some of the methods are prone to learn the model of the background as a consequence of tracking drift [4]. Moreover the object is usually represented as an axis aligned bounding box, therefore not adapting to changes in rotation or scale of the object [4], [5]. Both these methods adopt a patch-based model where the appearance is described using either the (normalized) patch itself, or some patch-based feature. In the spirit of more recent works [3], [16] we instead believe that an object should be described using a part based model, making it more flexible and robust to changes in appearance.

One problem facing online approaches is that the computations need to be performed in real time. The more appearances are learned, the longer it takes for the detection algorithm [5]. In [5] they solve this by restricting the number of appearances in the model, and removing old appearances if necessary. In the proposed model the number of appearances that needs to be learned is naturally bounded by the six sides of the inscribed cuboid, making it unnecessary to forget those previously learned.

III. Method

The tracking problem is described as the one of following an object identified in the first frame I_1 of an RGBD image sequence, through the remainder $I_2, ..., I_n$ of the sequence. We define the object as being the 3D structure corresponding to an *inscribed cuboid*, IC. The first cuboid IC_1 is manually defined in I_1 and then in every consecutive frame I_i, the position and orientation of IC_i is estimated, producing the sequence $IC_2, ..., IC_n$ as well as their 2D correspondences in the images, $b_2, ..., b_n$, as the tracking result.

This estimation is enabled through tracking of 2D keypoints p^j in the corresponding RGB image sequence. The tracking is done through both optical flow based tracking as well as feature based matching. Each point p^j has a corresponding 3D point P^j generated from the RGBD-data. The relation between the points P_1^j and the center of the cuboid, C_1, is known, either as a consequence of the initialization (Sec. III-A) or as a consequence of learning (Sec. III-H). Consequently, in a subsequent frame i the position and orientation of IC_i can be computed (Sec. III-C).

The following notation is used below:

- A *frame* $I_i = (RGB_i, RGBD_i)$ consists of an RGB image and a corresponding point cloud.
- The inscribed cuboid IC with corresponding centroid C
- $V_i \subset \{F, L, R, Ba, T, Bo\}$ indicates the visible faces of the cuboid (front, back, left, right, back, top, bottom) that are visible at frame i.
- An *observation* $\pi_i = [\pi_i^j] = ([p_i^j], [P_i^j])$ is an array of keypoints with attached feature descriptors p_i and corresponding 3D coordinates P_i.
- The *absolute object model* $\Pi = (\pi, IC, C)$. It consists of observations π, along with the defined inscribed cuboid IC with centroid $C = C_1$. π is initialized as π_i, and more observations are added as they appear in later frames i. Π^{V_i} corresponds to the subset of points that are visible on the faces V_i. The relative positions of the feature points with respect to the centroid are defined as $rm = [rm^j] = [P^j - C]$.
- The *background model* Π_B consists of an array of keypoints with feature descriptors.

Input: $I_1, ..., I_n, b_1$
Result: $b_2, ..., b_n$, $IC_2, ..., IC_n$
$IC_1 \leftarrow$ init_cube(b_1);
$\pi_1 \leftarrow$ extract_points(I_1);
$\Pi \leftarrow$ label_points(π_i, b_1);
$V \leftarrow F$;
for $i \in 2 : n$ **do**
  $\tilde{\pi}_i \leftarrow$ track_points(I_i, I_{i-1}, π_{i-1});
  $R_i \leftarrow$ get_rotation($\tilde{\pi}_i, \Pi$);
  $v_c \leftarrow$ vote($\tilde{\pi}_i, R_i, \Pi$);
  $C_i \leftarrow$ dbscan($v_c, \varepsilon, \lambda$);
  $\{IC_i, b_i\} \leftarrow$ update_cuboid(IC_{i-1}, C_i, R);
  $\hat{\pi}_i \leftarrow$ extract_points(I_i);
  $\hat{\pi}_i^m \leftarrow$ match_points($\Pi^V, \hat{\pi}_i$);
  $\pi_i \leftarrow$ update_point_set($\tilde{\pi}_i, \hat{\pi}_i^m$);
  $\tilde{V} \leftarrow$ get_visibility(R, C);
  $\Pi \leftarrow$ learn($\Pi, \hat{\pi}_i \backslash \hat{\pi}_i^m, \tilde{V}, R_i$);
  $V \leftarrow$ pick_visible($\tilde{V}$);
end

Algorithm 1: Overview of the algorithm.

- Subscript i refers to the frame number, superscript j refers to the index of a point. If only the subscript is used the variable indicates an array.

A. Initialization

The tracker is initialized with a manually defined bounding box b_1 in the first frame I_1. The *inscribed cuboid* IC_1 is initialized from b_1 with depth equal to the minimum of the width and height of b_1. Keypoints are extracted and BRISK features detected in the initial frame [17], and marked as *foreground* if they are within the input bounding box, *background* otherwise. An *absolute model* Π of the cuboid is constructed from IC_1 and keypoints belonging to the foreground, π_1. Its centroid is centered at the origin and π_1 are paired to its front face. The relative distance *rm* of each keypoint from the centroid of the cuboid is calculated. The front face is labeled as *learned*. A normal vector is initialized for each face of the absolute model according to Eq. (1).

$$\begin{aligned} \mu_{front} = [0,0,1]^T, \mu_{back} = [0,0-1]^T, \mu_{left} = [-1,0,0]^T \\ \mu_{right} = [1,0,0]^T, \ \mu_{top} = [0,-1,0]^T, \ \mu_{bottom} = [0,1,0]^T \\ N = \left[\mu_{front} \ \mu_{back} \ \mu_{left} \ \mu_{right} \ \mu_{top} \ \mu_{bottom}\right] \end{aligned} \quad (1)$$

The keypoints of the absolute model are used to calculate the transformation matrix as described in Sec. III-C. The matrix of normals N is used by the algorithm to detect which faces are visible to the camera, pick the corresponding points to track the object and learn new appearances of the object as explained in III-H. We also learn the model of the background Π_B since it helps to remove outliers in the matching step as in Sec. III-G.

B. Tracking

We track each keypoint in π_{i-1} by computing the movement from I_{i-1} to I_i by using the Lucas-Kanade pyramidal optical flow [18]. The estimated displaced points π_i^* are used to compute the backward points π_{i-1}^* from I_i to I_{i-1}. Keypoints in π_{i-1} that failed to be tracked, or have a Euclidean distance from their respective π_{i-1}^* higher that a defined threshold d are removed. Let $\tilde{\pi}_i$ be the remaining points in π_i^*, that were backprojected correctly.

C. Estimating Rotation

The set of tracked keypoints $\tilde{\pi}_i = (\tilde{p}_i, \tilde{P}_i)$ is used to estimate the rotation of the object from its starting position in $RGBD_1$ using the set of points $\tilde{P}_1$. The rotation matrix is calculated using the following steps. First the centroids of P_1 and $\tilde{P}_i$ are computed:

$$\eta_1 = \frac{1}{N}\sum_{j=1}^{N} P_1^j, \eta_i = \frac{1}{N}\sum_{j=1}^{N} \tilde{P}_i^j \quad (2)$$

(Note that this is not the same as the centroid C_i.)

The $3x3$ covariance matrix H is calculated by accumulating the product of $3D$ points translated to the origin:

$$H = \sum_{j=1}^{N} \left(P_1^j - \eta_1\right)\left(\tilde{P}_i^j - \eta_i\right)^T \quad (3)$$

The singular value decomposition is then used to compute the rotation matrix R from H.

$$[U, S, V] = SVD(H) \quad (4)$$

$$R_i = VU^T \quad (5)$$

D. Voting for Centroid

Each valid tracked point in the current frame votes for the updated centroid position of the object. The voting is performed by using the relative positions rm_1 of the points in our absolute model and the current estimated position of the points $\tilde{P}_i$:

$$v_c^j = \tilde{P}_i^j - R_i * rm_1^j \quad (6)$$

The set of votes v_c corresponds to the translation from the absolute model to the current estimate for each of the tracked keypoints. An illustration of this can be found in Fig. 2.

E. Clustering Votes

Given the set of centroid votes v_c we estimate the updated centroid position C_i by finding a dense cluster of votes and filter outliers. This is performed using the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [19]. The algorithm allows to detect clusters of arbitrary shape and does not require the number of clusters to be set beforehand. It can separate points that belongs to a dense cluster, points that are connected to a densely populated area but that lies on its border, and outliers. It requires two parameters:

- ε, which defines the maximum distance to consider two data points mutually reachable
- λ, which defines the minimum number of points to form a cluster.

Each vote $v^k \in v_c$ is considered *reachable* from v^j if they are within Euclidean distance ε from each other. Moreover if

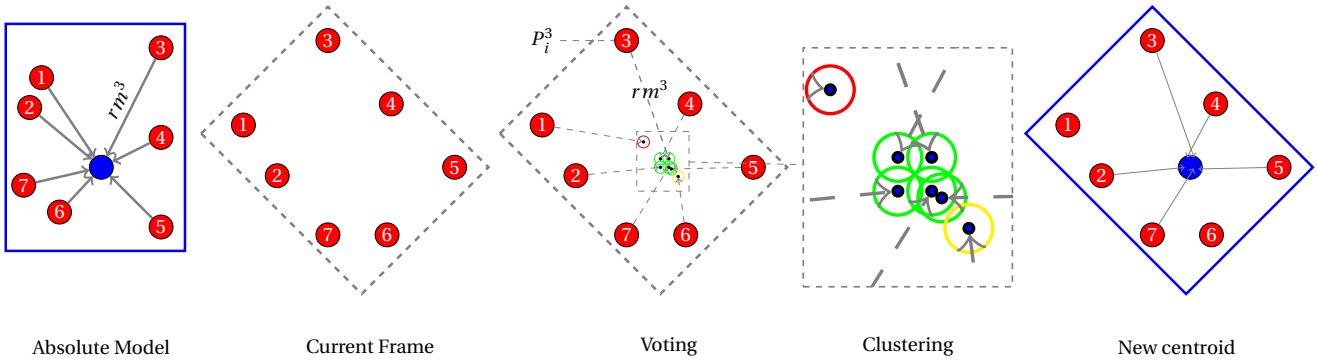


Fig. 2. Explanation of the tracking part of the method. The keypoints tracked in the current frame and their counterparts in the absolute model are used to estimate the rotation matrix, the relative distance of each point is used to vote for the new centroid position. DBScan is used to filter nodes on the border of a cluster (shown in yellow) and outliers (shown in red). The remaining votes are used to compute the new centroid position and the cuboid is centered accordingly. The image shows an example in 2D for simplification but the implemented framework works in 3D.

v^j is *reachable* from a sufficient number of votes λ, all votes within the *reachable* range of v^j are considered part of the same cluster. In order to compute the new centroid C_i of the object we choose the cluster with the higher density of points. Fig. 2 shows an example of points that are contributing to the computation of the centroid. Points in green are considered part of the chosen cluster while points in yellow are the ones consider o the *border* of the cluster. The overall complexity of the algorithm is $O(N \log N)$.

F. Updating Centroid and Inscribed Cuboid

The updated centroid C_i position is obtained by computing the mean of the chosen cluster in the previous step (Fig. 2). The *inscribed cuboid* pose (Fig. 3) is estimated by using the updated centroid, the relative distance of each point of the cuboid from the centroid in the absolute model and the estimated rotation matrix using the following equation:

$$P_i = C_i + R_i * rm \tag{7}$$

G. Matching Points

We extract BRISK features from each input frame, such features are used as a recovery mechanism for keypoints belonging to the object that have been lost by the tracker or to detect the object again. We exhaustively match each keypoint p_i^j extracted at time i with the descriptors of the faces of the object that are currently visible and used for tracking. As the keypoint descriptors are binary we compute the hamming distance between all pairs of keypoints. A keypoint is matched to its nearest neighbor if their distance is lower that a manually defined threshold ϕ and if the ratio between the nearest neighbor and the second nearest neighbor is lower than ρ.

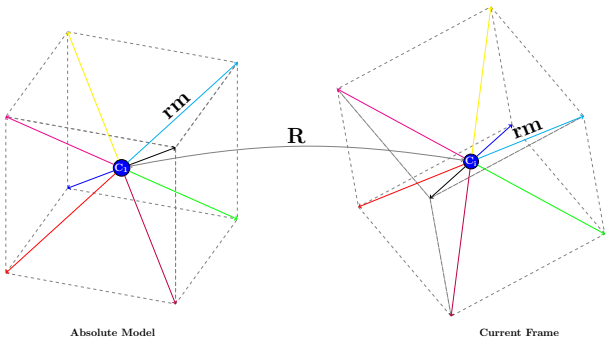


Fig. 3. Each vertex of the cuboid is represented as the relative distance vector from the centroid of the absolute model. The updated cuboid is then computed by rotating the vectors using the estimated transformation R as explained in Eq. 7.

H. Learning and Recovery

We develop a mechanism to evaluate the current estimation of the object pose to chose the learned feature points to use in tracking and to learn new appearances of the model. The mechanism takes into account 4 aspects:

Visibility: Given the matrix of normals N (Sec. III-A) and the rotation matrix R it is possible to determine which faces of the inscribed cuboids are currently visible from the camera. Multiplying the normal matrix by the last column of the rotation matrix, we obtain the visibility vector $V_{vs} = N * R[:,2] = 1x6$ where each element has a value in the interval $[-1,1]$. A negative value indicates that the face is facing away from the camera, and the bigger number, the closer the face is to being parallel to the image plane. Since a keypoint learned from a surface parallel to the camera is less sensitive to rotations, our method learns a new appearance only if a face is visible more that a threshold κ, while it uses the keypoints of a learned face only if the visibility is greater that ι. Fig. 4 shows an example of the computation where initially only one face is visible to the camera having its visibility value close to 1, however its visibility decrease as soon as the object rotates, while another face visibility increases.

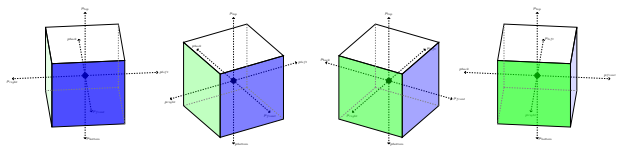


Fig. 4. In the first image the front face is facing the camera, while the right face is barely visible. As the object rotates the right face becomes more visible while the visibility of the front face decreases.

Stability and Motion: An incorrect estimation of the cuboid pose may result in the partial inclusion of the background in the object model. It is essential to detect these scenarios, often occurring due to outliers in the estimation of R, to prevent the learning of an incorrect appearance. To detect such scenarios we extract the quaternion q_i from R_i at each time step and compute the median angular distance quaternions with the quaternions computed in the last t frames. Moreover matching features is essential as a recovery procedure when the tracker loses the object. However, since features are sensitive to motion of the object, it is important to extract features from the object appearance in situations when the object motion is close to 0. Therefore the computation of a low median angular distance ΔQ indicates a robust estimated pose and slow motion of the object while a peak in the signature of the quaternions indicates a noisy frame. The method learns only if the value is lower than a certain treshold γ.

History: Given a new appearance candidate face to learn F^i_{new}, it is learned if the quality of the new appearance is higher that the current learned face F^i_{old}. This is performed by comparing the quality of the new candidate with the learned model in terms of visibility, stability and motion as in Eq. 8.

$$F^i = \begin{cases} F^i_{new} \; if \; V^i_{new} > V^i_{old} \wedge \Delta Q^i_{new} < \Delta Q^i_{old} \\ F^i_{old} \; otherwise \end{cases} \tag{8}$$

I. Backprojection

Given a keypoint π^j_i detected at time i we want to back project it and incorporate it to the absolute model as shown in Fig. 5. This is achieved using the following equation:

$$rm^j_1 = R^{-1}_i (P^j_i - C_i) \tag{9}$$

where R_i is the estimated rotation matrix at time i and C_i is the estimated object centroid. rm is the relative distance of the keypoint from the object centroid in the first frame, therefore the absolute position of point p_{k_1} is:

$$P^j_1 = rm^j_1 + C_1 \tag{10}$$

Fig. 6 shows examples of keypoints extracted from new appearance candidates that are projected back to the initial

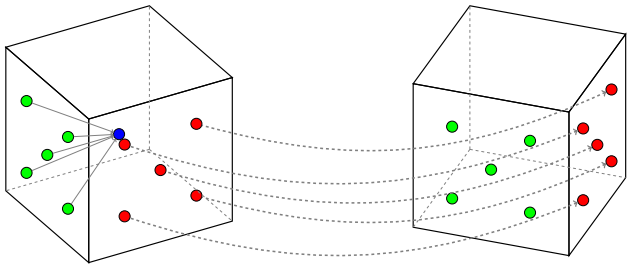


Fig. 5. Points in green are currently use to calculate the transformation matrix from the ideal model and estimate the new object pose. Points in red are extracted from a new face visible to the camera and back projected to the ideal model as part of the learning procedure.

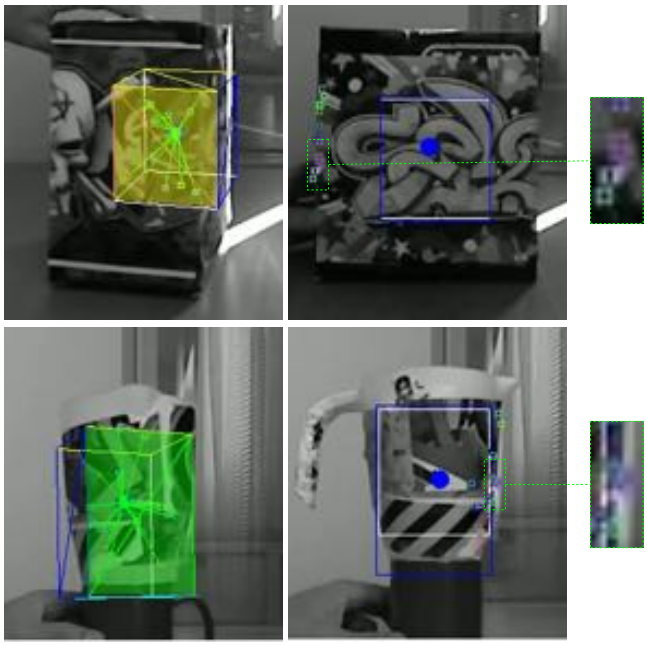


Fig. 6. Examples of back projection of points using arbitrary shape objects. The column on the left shows the tracked objects in the current frame and the learned points that are projected to the absolute model used for pose estimation. It can be noticed that the points are projected correctly since the real depth is used and the inscribed cuboid just delimit the area where they can be learned from.

frame of the sequence. It can be noticed that while the shape of the object is drastically different from the *inscribed cuboid*, the keypoints extracted from the new learned face are still projected correctly on the surface of the object in the absolute model. This allows the tracker to be flexible to object shape since the cuboid determines just the area from where the keypoints can be extracted to prevent the learning of the background.

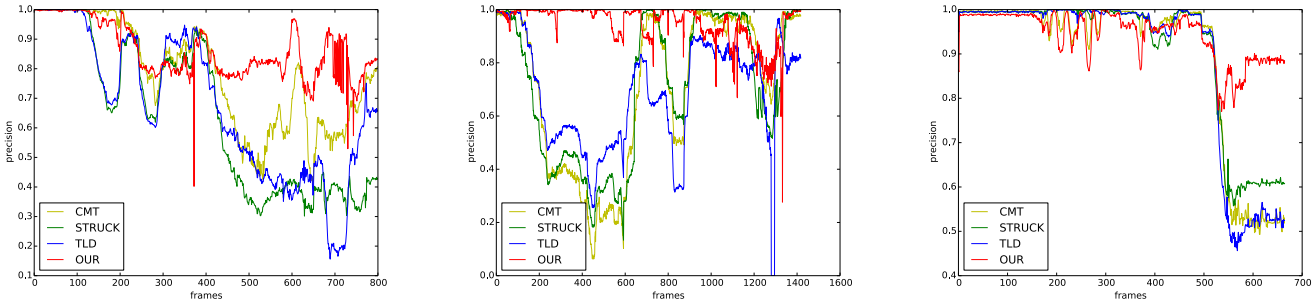
J. Recovery

In case the object is totally lost the matcher compare the features extracted from the current frame with all the features of the learned faces so far and pick as a visible face the one with the most matches if it exists.

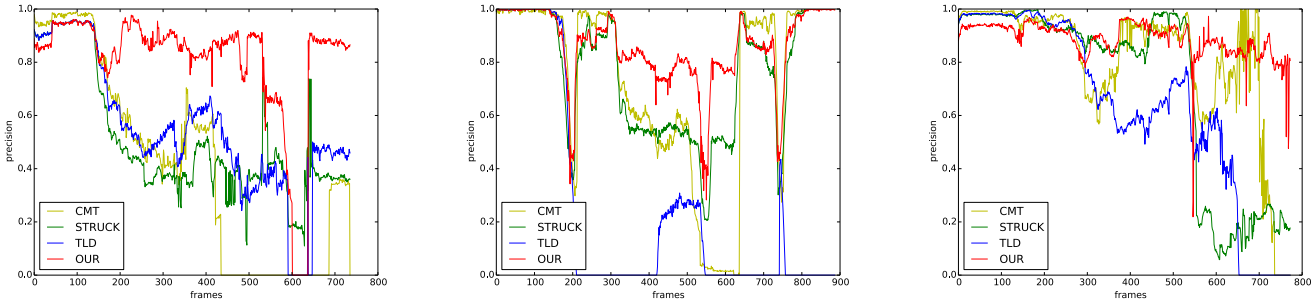
IV. Experiments

A. Parameter settings

The tracking step uses the Euclidean distance $d = 10$ between the forward computation and the backward computation to decide if a tracked point is lost. The clustering algorithm DBScan uses $\varepsilon = 10$ to search for density reachable points and $\lambda = 5$ to indicates the minimum number of neighbors a point requires in order to be considered a potential cluster center. A new face should have a visibility value $\kappa > 0.5$ to be taken into account as a learning candidate and $\iota > 0.6$ to use a learned face for tracking. Moreover the median quaternion distance γ to measure the quality of the current estimated model should be lower than 10 degrees. The matcher uses $\phi > 0.75$ and $\rho < 0.8$.



(a) In-plane rotation (b) Out-of-plane rotation (c) In-plane and out-of-plane rotation



(d) Out-of-plane rotation and occusion (e) Occlusion and Out-of-plane rotation (f) Hand manipulation

Fig. 7. Precision of the trackers on a subset of the videos used for experiments. See legend for tracker color coding. It can be seen that the proposed method (red) is more robust, performing at par with, or outperforming the other trackers.

B. Benchmark

We perform qualitative experiments to test the robustness of the tracker against other state of the art trackers. We picked the trackers that performed the best in the extensive evaluation presented in the [10]. The trackers used for our evaluation are: Structured Output Tracking with Kernels (STRUCK) [5], Tracking-Learning-Detection (TLD) [4] and Consensus-based Matching and Tracking of Keypoints (CMT) [3]. The first two learn new appearances of the object to track but as they may learn the background they suffer from drifting. Moreover they do not deal with object rotation. CMT is more precise as it deals with rotation but it does not learn new appearances so it fails when the object appearance changes. The code of all trackers has been found online and they are run using the default parameter setting. Since our main goal consists in deploying a tracking algorithm that can track manipulable objects we recorded a set of **20** videos where the objects are subject to fast movements, rotation, change in appearance and occlusion. We labeled each video with the mask of the object in a supervised way using an external tool [20]. We compute the precision (Fig. 7) and recall (Fig. 8) of each tracker given the ground truth. The average performance is shown in Table I. The results shown are taken from a subset of the videos used for the experiments that summarizes the problems to tackle: movement, rotation and occlusion. It can be noticed that upon simple movement the performance of the trackers is similar, however as there is a rotation or the speed increases [5] and [4] drift as they start to learn the background. [3] is more reliable than the others upon in plane rotations but it is the worst when there are changes in appearance since it does not learn any new appearances of the tracked object. Some interesting behaviors can be noticed in Fig. 9: (e) shows that upon partial occlusion TLD learns the occluding object as the object to track, moreover CMT loses totally the object upon a partial out of plane rotation and occlusion. (f) shows that during the manipulation of an object TLD and STRUCK learn the hand as the object model even if the hands are not occluding the object broadly.

TABLE I

AVERAGE RESULTS COMPUTED USING THE SET OF VIDEOS RECORDED FOR THE EXPERIMENTS.

	CMT	TLD	STRUCK	Our
Precision	0.69	0.68	0.61	0.94
Recall	0.46	0.52	0.40	0.76

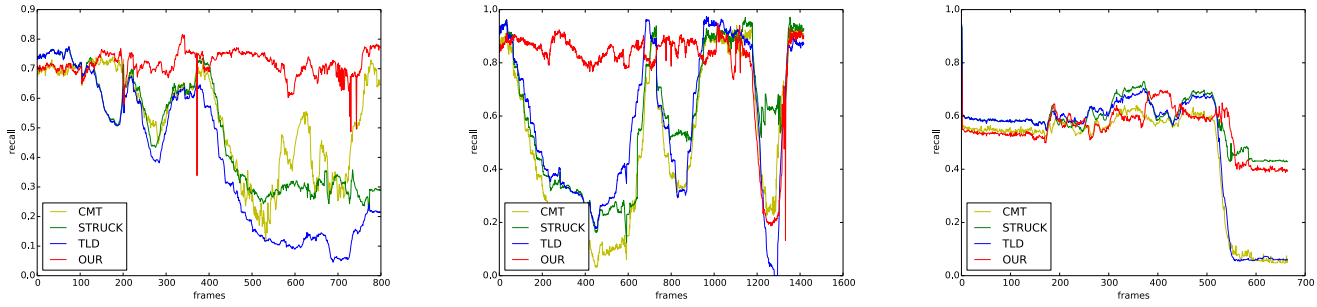
C. Performance

The current implementation of the algorithm is in $C++$ without any form of parallelism but using vector instrinsics since BRISK features are binary. The algorithm has been tested on an desktop computer with an Intel core $i74930K$ and a laptop with with Intel core $i73687U$ achieving an average frame rate of 70 FPS on the desktop and 55 FPS on the laptop. It can be noticed in Table II that the most time consuming operations are the extraction of the features and

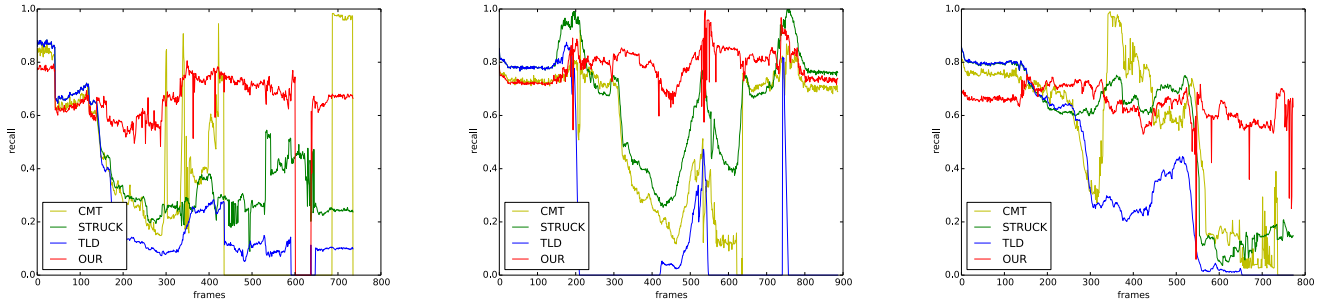
TABLE II

IMPACT OF THE DIFFERENT STEPS ON THE PERFORMANCE. AVERAGE NUMBER OF KEYPOINTS TRACKED BETWEEN 50-100 AND KEYPOINTS EXTRACTED AROUND 300.

	Tracking	Voting	Clustering	Features	Matching	Visibility	Others	Tot
Ms	3.64	0.3	0.10	4.40	2.40	0.3	3.0	14.16
%	0.25	0.02	0.00	0.31	0.17	0.02	0.21	1.00



(a) In-plane rotation (b) Out-of-plane rotation (c) In-plane and out-of-plane rotation



(d) Out-of-plane rotation and occusion (e) Occlusion and Out-of-plane rotation (f) Hand manipulation

Fig. 8. Recall of the trackers on a subset of the videos used for experiments. See legend for tracker color coding. It can be seen that the proposed method (red) is more robust, performing at par with, or outperforming the other trackers. Only in one scenario, (b) frames 1200-1400, the STRUCK tracker (green) has a significantly higher recall than the proposed method due to a pose estimation error (see the tracking results in Figure 8(b), frame 4).

the calculation of optical flow, operations that can be highly parallelized. Moreover since the feature matcher contribution is important as a recovery mechanism upon tracking failures the two mechanisms can be run concurrently on separate threads. The code of the tracker will be released publicly.

V. Conclusions and Future Work

In this paper we present a framework to detect, learn and track unknown 3D objects. The method is a natural extension of a 2D bounding box tracker. We present results that indicate the benefit of the method compared to other state-of-the-art tracker. The use of the inscribed cuboid prevents drifting while tracking and bounds the complexity of the learning procedure since the required appearances to learn and remember are limited by the number of faces of the cuboid.

Since the method tracks the centroid of unknown objects, a natural improvement consists in segmenting the object shape from the background given the centroid and build an approximation of the $3D$ object shape. This will allow to fill the gap with a model based tracker. It will be interesting also to learn the texture of the object and used the approximated shape and texture to generate simulated results that can be compared with the real scenario and improve the stability of the estimated optical flow as proposed by a state-of-the-art model based tracker [6].

References

[1] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen. Categorizing object-action relations from semantic scene graphs. In *ICRA*, 2010.

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

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

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

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

[6] K. Pauwels, L. Rubio, J. Diaz, and E. Ros. Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues. In *CVPR*, 2013.

[7] C. Bibby and I. Reid. Robust real-time visual tracking using pixel-wise posteriors. In *ECCV*, 2008.

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

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

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

[11] D. Mitzel and B. Leibe. Taking mobile multi-object tracking to the next level: People, unknown objects, and carried items. In *ECCV*, 2012.

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

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

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

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

[16] S. Hare, A Saffari, and P.H.S. Torr. Efficient online structured output learning for keypoint-based object tracking. In *CVPR*, 2012.

[17] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints. *ICCV*, November 2011.

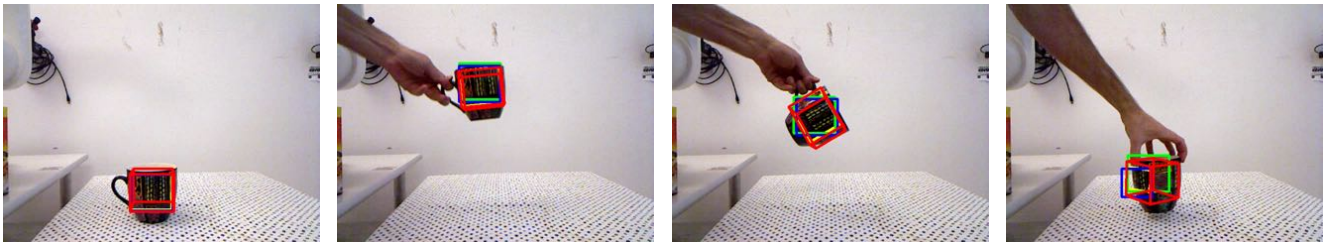
[18] J-Y. Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker, 2000.

[19] M. Ester, H-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[20] P. Bertolino. Sensarea: An authoring tool to create accurate clickable videos. In *CBMI*, 2012.

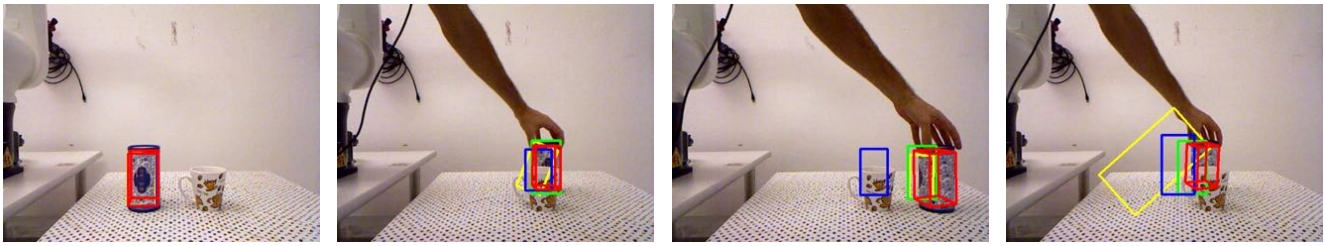
(a) In-plane rotation (frame 3)

(b) Fast motion (frame 2) and out-of-plane rotation (frame 3, 4)

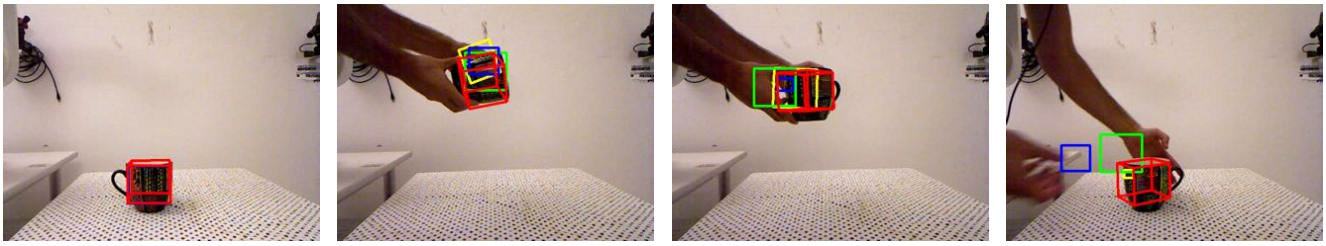


(c) In-plane rotation (frame 3) and out-of-plane rotation (frame 4)

(d) Out-of-plane rotation (frame 2) and occlusion (frame 4)



(e) Occlusion (frame 2,4), Out-of-plane rotation (frame 3)



(f) Hand manipulation (frame 2,3)

Fig. 9. Results achieved using state of the art trackers on objects subject to rotations and change in appearance. Each color correspond to a tracker: green to STRUCK, yellow to CMT, blue to TLD while red is the result achieved using our tracker. The sequences shown here are a subset of the videos used for experiments. They summarize the problems an object tracker has to address: motion, rotation, occlusion. It is interesting to see in the last figure how the trackers that learn drift away from the object as they incorporate the hand appearance in the model during the manipulation activity.