A Versatile Learning-based 3D Temporal Tracker: Scalable, Robust, Online

David Joseph Tan[1], Federico Tombari[1,2], Slobodan Ilic[1,3], Nassir Navab[1]
[1]CAMP, Technische Universität München
[2]DISI, Università di Bologna
[3]Siemens AG
{tanda,tombari,ilics,navab}@in.tum.de, federico.tombari@unibo.it, slobodan.ilic@siemens.com

Abstract

This paper proposes a temporal tracking algorithm based on Random Forest that uses depth images to estimate and track the 3D pose of a rigid object in real-time. Compared to the state of the art aimed at the same goal, our algorithm holds important attributes such as high robustness against holes and occlusion, low computational cost of both learning and tracking stages, and low memory consumption. These are obtained (a) by a novel formulation of the learning strategy, based on a dense sampling of the camera viewpoints and learning independent trees from a single image for each camera view; as well as, (b) by an insightful occlusion handling strategy that enforces the forest to recognize the object's local and global structures. Due to these attributes, we report state-of-the-art tracking accuracy on benchmark datasets, and accomplish remarkable scalability with the number of targets, being able to simultaneously track the pose of over a hundred objects at 30 fps with an off-the-shelf CPU. In addition, the fast learning time enables us to extend our algorithm as a robust online tracker for model-free 3D objects under different viewpoints and appearance changes as demonstrated by the experiments.

1. Introduction

This paper focuses on 3D object temporal tracking to find the pose of a rigid object, with three degrees of freedom for rotation and three for translation, across a series of depth frames in a video sequence. Contrary to *tracking-by-detection*, which assumes the frames independent from each other such that it detects the object and estimate its pose in each frame, *object temporal tracking* relies on the transformation parameters at the previous frame to estimate the pose at the current frame. Theoretically, by temporally relaying the transformation parameters, it localizes the object within the frame instead of re-detecting the object, and it only requires to estimate the changes in the transformation parameters from one frame to the next instead of searching in the entire pose space. As a result, most object temporal tracking algorithms are significantly faster than tracking-by-detection. Real-time 3D tracking is now the enabling technology for a range of applications in the field of augmented reality, robotic perception as well as human-machine interaction. In these cases, tracking multiple objects at real-time becomes an inherent requirement. Moreover, when using 3D sensors on mobile devices, low computational cost and memory consumption are required.

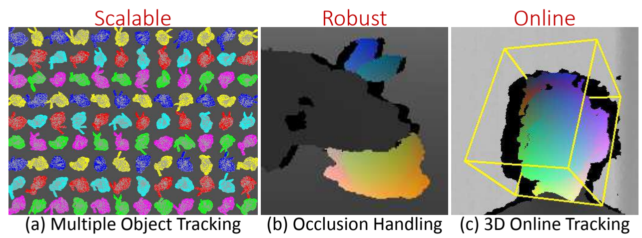


Figure 1. Qualitative evaluations for (a) multiple object tracking, (b) occlusion handling and (c) 3D online tracking for head pose estimation. More evaluations are in the *Supplementary Materials*.

In this paper, we propose a general 3D real-time object temporal tracking algorithm from depth images that is inherently scalable, being able to track more than a hundred 3D objects at 30 fps (Fig. 1(a)), and robust to high levels of occlusions (Fig. 1(b)). Our approach is versatile to be used for both model-based as well as online model-free tracking, where the tracker initializes the geometry of the target from a single depth frame and adapts it to changing geometry and unseen camera viewpoints while tracking (Fig. 1(c)).

More specifically, to achieve generalization, our tracker requires the following attributes in order to satisfy the diverse requirements of most 3D tracking applications:

(1) *Robustness.* To avoid tracking failures, the tracker must be robust against sensor noise such as holes and artifacts, commonly present in depth data, as well as robust against partial occlusion from the environment.

(2) *Tracking time and computational cost.* Due to the theoretical efficiency, the tracker must be faster than any tracking-by-detection method. Moreover, it must spec-

ify the computational cost to attain this speed.

(3) *Memory consumption.* The amount of memory the tracker consumes from RAM for a single target should be small enough to allow simultaneous tracking of multiple targets along the same sequence.

(4) *Scalability to multiple objects.* An increase in the number of simultaneously tracked objects causes an increase in (2) tracking time and computational cost, and (3) memory consumption in comparison to tracking a single object. Moreover, it emphasizes how additional objects affect the (1) robustness of the algorithm.

In addition, for all learning-based methods, it is also essential to consider the:

(5) *Learning time.* This includes the creation of the learning dataset from loading or rendering images to extracting the input (samples) and output (labels) parameters, and the construction of the machine learning data structure. It is particularly important for online tracking, where the object has to be incrementally learned in the successive frames at real-time.

Therefore, the novelty of the work is that it satisfies all of the aforementioned attributes simultaneously, while achieving better results against the other methods individually. Notably, we evaluate our tracker in Sec. 3 based on them, achieving state of the art results.

Our method is inspired by the learning-based approach of [23] that uses depth images only. It is a temporal tracking algorithm based on Random Forest [4] that runs at 2 ms per frame with a single CPU core. At the moment, this is the only method that has achieved this efficiency in 3D tracking with an extremely low computational requirement compared to the literature concerning 3D tracking-by-detection [2, 7, 11] and 3D temporal tracking [3, 5]. However, it poses problems in the robustness against large occlusions and large holes that results in tracking errors and failures, memory consumption that limits tracking to a maximum of 9 objects and long learning time that limits its applicability to model-based tracking.

Hence, in contrast to [23], our proposed tracker overcomes these problems through an algorithm that (1) is more robust to holes and partial occlusions, (3) has a very low memory footprint, (4) is scalable to track a hundred objects in real-time and (5) has a fast learning time, while keeping the existing attributes regarding (2) low tracking time with a low computational expense.

Our main theoretical contribution is two-fold. On one hand, we propose a novel occlusion handling strategy that adapts the choice of the input samples being learned. In effect, this notably increases the overall robustness, as proven through the state-of-the-art results reported by our method on benchmark datasets (see Sec. 3). On the other hand, in lieu of the learning strategy employed by [23], we propose to use only one depth image to create the entire learning dataset, or the entire set of samples and labels for each camera view. This leads to a novel formulation of the learning strategy, which allows a much denser sampling of the camera viewpoints with respect to [23]. As a consequence, we achieve not only a high scalability, but also a remarkably low memory footprint and fast learning time, that allows our proposal to be deployed in an online 3D tracking context, which initializes the geometry of the target from a single depth frame, and adapts it to changing geometry and unseen camera viewpoints while tracking.

Related works. If we limit our scope to temporal trackers that estimate the object's pose using solely depth images, there are only two existing methods — the energy-minimization such as Iterative Closest Point (ICP) algorithms [3, 5] and a learning-based algorithm [23]. Most works [2, 11, 12, 15] have applied ICP as an integral component of their algorithms; while, others [9, 16, 19, 22] have developed it to different extensions. Nonetheless, to the best of our knowledge, there have been only one learning-based object temporal tracking algorithm that relies solely on depth images [23].

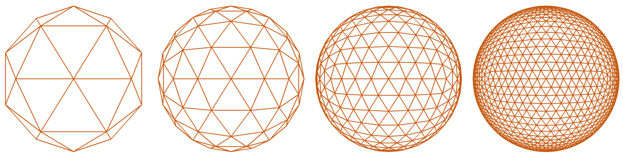
Furthermore, there are several works that have utilized the RGB-D data. This includes the hand-held object tracking [10] that uses RGB to remove the hand before running ICP. Moreover, the particle filter approaches [6, 13] extends existing RGB trackers to include the depth data. Another work [17] uses level-set optimization with appearance and physical constraints to handle occlusions from interacting objects; but, they only conduct their experiments on textureless objects with simple geometric structure such as prisms or spheres. Among the RGB-D methods [6, 10, 13, 17], it is common to implement them in GPU for real-time tracking. In effect, their runtime depends on the type of GPU that they use, which creates a problem to track more objects while still keeping the real-time performance.

2. Object temporal tracking

Tracking aims at solving the registration problem between the 3D points on the object and the 3D points from the depth image representing the current frame. To register these two sets of points, the error function is defined as the signed displacement of a point correspondence:

$$\epsilon_j^v(\mathbf{T};\mathbf{D}) = \mathbf{N}_v \cdot \left(\mathbf{T}^{-1}\mathcal{D}(\mathbf{x}_j) - \mathbf{X}_j\right) \tag{1}$$

where $\mathbf{X}_j$ is a point on the object in the object coordinate system, $\mathbf{N}_v$ is a unit vector that defines the direction of the displacement (see Eq. 3), $\mathbf{T}$ is the object transformation from the camera (see Eq. 2), $\mathbf{x}_j$ is the projection of $\mathbf{T}\mathbf{X}_j$, and $\mathbf{D}$ is the depth image with $\mathcal{D}(\mathbf{x})$ as the back-projection function of the pixel $\mathbf{x}$. As notations, we include a tilde as $\tilde{\mathbf{x}}$ to denote inhomogeneous coordinates while $\mathbf{x}$ as homogeneous.



(a) 42 (b) 162 (c) 642 (d) 2562

Figure 2. The geodesic grids, which locate the camera around the target object, are derived from recursively dividing an icosahedron with 12 vertices to (a) 42, (b) 162, (c) 642 and (d) 2562 vertices.

The objective of tracking is to locate the object in the image by finding the transformation that registers the points on the object to the points from the depth image. Specifically, object temporal trackers seek the transformation $\mathbf{T}_t$ from the frames at time $t-1$ to t and transform $\mathbf{X}_j$ by $\hat{\mathbf{T}}_t = \prod_{i=0}^{t} \mathbf{T}_i$. In the current frame $\mathbf{D}_t$, it utilizes the displacement of the points $\epsilon_j^v(\hat{\mathbf{T}}_{t-1}; \mathbf{D}_t)$ to determine the relative transformation $\mathbf{T}_t$ that minimizes $\epsilon_j^v(\hat{\mathbf{T}}_t; \mathbf{D}_t)$.

Instead of aggregating the errors as $\sum_j |\epsilon_j^v|^2$ in energy minimizers, we take the individual values of the signed displacements $\epsilon_j^v(\hat{\mathbf{T}}_{t-1}; \mathbf{D}_t)$ from n_j points on the object $\{\mathbf{X}_j\}_{j=1}^{n_j}$ as the input to the Random Forest [4] and predict the transformation parameters of $\mathbf{T}_t$. However, similar to energy minimizers, the tracker runs several iterations on each frame to refine the predicted pose.

Parametrization. The rigid transfomation $\mathbf{T}$ is constructed with the Euler angles α, β and γ, and the translation vector $\tilde{\mathbf{t}} = (t_x, t_y, t_z)^\top$ such that:

$$\mathbf{T} = \mathbf{R}(\alpha, \beta, \gamma) \cdot \begin{bmatrix} \mathbf{I}_{3\times 3} & \tilde{\mathbf{t}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{2}$$

with the parameter vector $\boldsymbol{\tau} = [\alpha, \beta, \gamma, \tilde{\mathbf{t}}^\top]^\top$.

Dense camera. When the object moves during tracking, its viewpoint changes and the visible points on the object also vary accordingly. Thus, to ensure the capacity to track the object from different viewpoints, the algorithm learns the relation between the error function and the transformation parameters from different viewpoints or camera views. It follows that, in tracking, the closest camera views have the highest similarity to the current frame and only the trees from these views are evaluated to predict the relative transformation.

For instance, in model-based tracking, n_v views of the object's model are synthetically rendered by positioning the camera on the vertices of a densely-sampled geodesic grid [21] around the object. This is created by recursively dividing an icosahedron into equally spaced n_v vertices, as shown in Fig. 2. By increasing n_v, the distance between neighboring cameras is decreased. In effect, the trees from multiple neighboring camera views predict the output parameters, instead of evaluating a number of trees from one view in [23]. Consequently, we can significantly decrease the number of trees per view in comparison to [23]. Thus, each view independently learns one tree per parameter using the corresponding rendered image. This produces a total $6n_v$ trees in the forest from all views.

Although one can argue to increase the number of camera views for [23], this is impractical because of the time required to generate the increased number of rendered images. As an example, when using 642 views in Fig. 2(c), they need a total of 32.1M images for the learning dataset from all camera views, while our method needs 642 images, *i.e.* one for each camera view.

Whether using synthetic or real depth images, the input to learning from one view is a depth image $\mathbf{D}_v$ and its corresponding object transformation $\mathbf{T}_v$. In the object coordinate system, the location of the camera $\tilde{\mathbf{X}}_v$ is:

$$\tilde{\mathbf{X}}_v = -\tilde{\mathbf{R}}_v^\top \tilde{\mathbf{t}}_v \;\Rightarrow\; \mathbf{N}_v = \left(\frac{\tilde{\mathbf{X}}_v^\top}{\|\tilde{\mathbf{X}}_v\|_2}, 0 \right)^\top \tag{3}$$

where $\tilde{\mathbf{R}}_v$ is the 3×3 rotation matrix and $\tilde{\mathbf{t}}_v$ is the translation vector of $\mathbf{T}_v$. From this, we define the unit vector $\mathbf{N}_v$ from Eq. 1 as the vector that points towards the camera center. Instead of the normal to the object's surface, the advantage of using Eq. 3 is evident with real depth images, where the normal to the object's surface becomes expensive to compute and prone to large errors due to sensor noise.

2.1. Learning from one viewpoint

While looking at the object from a given viewpoint v, the depth image $\mathbf{D}_v$ and the corresponding object transformation $\mathbf{T}_v$ are taken as the input to learning. Using $\mathbf{D}_v$ and $\mathbf{T}_v$ from only one view of the object, the visible points on the object are extracted to create the learning dataset and, eventually, learn the trees. Among the pixels $\{\mathbf{x}_i\}_{i=1}^{n_i}$ from $\mathbf{D}_v$ that are on the object, n_j points are selected, back-projected and transformed to the object coordinate system. These are the set of points on the object $\boldsymbol{\chi}_v = \{\mathbf{X}_j\}_{j=1}^{n_j}$ that are used to compute the displacements in Eq. 1. As a consequence, we are tracking the location of $\boldsymbol{\chi}_v$ across time by transforming them with $\hat{\mathbf{T}}_t$.

Occlusion handling. Even though randomly selecting a subset of points on the object endows the tracker with robustness against small holes on the depth image [23], occlusions still affect its performance. By observation, we describe an occlusion on an image as a 2D obstruction that covers a portion of an object starting from an edge of the object's silhouette, while the other regions are visible to the camera, as demonstrated in Fig. 3. Using this observation, the object on the image are divided into two regions using a line with a unit normal vector $\mathbf{n}_l$, where the pixels from one region is selected for $\boldsymbol{\chi}_v$ and $\mathbf{n}_l$ is a random unit vector within the 2π unit circle. Thereupon, the pixels are sorted based on $d_i = \mathbf{n}_l \cdot \mathbf{x}_i$ such that the pixels with a lower value are located on one edge of the object while the pixels with a higher value are on the opposite edge. Hence, only the first 10% to 70% of the sorted pixels are included for the selection of $\boldsymbol{\chi}_v$, where we randomly choose the percentage of pixels. In effect, occlusion is handled by discarding a subregion of the object and selecting the set of points $\boldsymbol{\chi}_v$ from the remaining subregion as illustrated in Fig. 3.

Dataset. To build the learning dataset from $\mathbf{D}_v$, $\mathbf{T}_v$ and $\boldsymbol{\chi}_v$, the rotation angles and translation vector in $\boldsymbol{\tau}_r$ of Eq. 2 are randomly parametrized to compose $\mathbf{T}_r$ and formulate $\hat{\mathbf{T}}_r = \mathbf{T}_v \mathbf{T}_r^{-1}$. By transforming $\mathbf{X}_j$ by $\hat{\mathbf{T}}_r$, it emulates the location of the points from the previous frame such that the current frame needs a transformation of $\mathbf{T}_r$ to correctly track the object. Consequently, $\hat{\mathbf{T}}_r$ is used to compute the displacement vector $\boldsymbol{\epsilon}_r^v = [\epsilon_j^v(\hat{\mathbf{T}}_r; \mathbf{D}_v)]_{j=1}^{n_j}$. After imposing n_r random parameters, the accumulation of $\boldsymbol{\epsilon}_r^v$ and $\boldsymbol{\tau}_r$ builds the learning dataset $\mathcal{S} = \{(\boldsymbol{\epsilon}_r^v, \boldsymbol{\tau}_r)\}_{r=1}^{n_r}$. In this way, the forest aims at learning the relation between $\boldsymbol{\epsilon}$ and $\boldsymbol{\tau}$; so that, when $\boldsymbol{\epsilon}$ is given in tracking, the forest can predict $\boldsymbol{\tau}$.

Learning. Given the dataset $\mathcal{S}$, learning aims at splitting $\mathcal{S}$ into two smaller subsets to be passed down to its children. The tree grows by iteratively splitting the inherited subset of the learning dataset $\mathcal{S}_N$ and passing down the resulting $\mathcal{S}_l$ and $\mathcal{S}_r$ to its left and right child. The objective is to split using $\boldsymbol{\epsilon}$ while optimizing a parameter in $\boldsymbol{\tau}$ to make the values more coherent which is measured by the standard deviation $\sigma(\mathcal{S})$ of the parameter from all $\boldsymbol{\tau}$ in $\mathcal{S}$.

To split $\mathcal{S}_N$ into $\mathcal{S}_l$ and $\mathcal{S}_r$, an element of the vector $\boldsymbol{\epsilon}$ across all $\mathcal{S}_N$ is thresholded such that all values that are less than the threshold goes to $\mathcal{S}_l$ while the others go to $\mathcal{S}_r$. All of the n_j elements of ϵ and several thresholds that are linearly space between the minimum and maximum values of the each element across $\mathcal{S}_N$ are tested to split the dataset. These tests are evaluated based on the information gain computed as:

$$G = \sigma(\mathcal{S}_N) - \sum_{i \in \{l,r\}} \frac{|\mathcal{S}_i|}{|\mathcal{S}_N|} \sigma(\mathcal{S}_i) \tag{4}$$

where the test with highest information gain gives the best split. As a result, the index of the element in the vector and the threshold that gives the best split are stored in the node.

The tree stops growing if the size of the inherited learning dataset is too small or the standard deviation of the parameter is less than a threshold. Then, this node is a leaf and stores the mean and standard deviation of the parameter.

Consequently, the same learning process is applied for each of the parameters in $\boldsymbol{\tau}$ to grow one tree per parameter. It is also applied to all of the n_v views of the object.

2.2. Tracking an object

When tracking an object at time t, the given input is the current frame $\mathbf{D}_t$, the object transformation from the previous frame $\hat{\mathbf{T}}_{t-1}$ and the learned forest with $6n_v$ trees. Ultimately, the forest predicts the parameters of $\mathbf{T}_t$ and updates the object transformation from $\hat{\mathbf{T}}_{t-1}$ to $\hat{\mathbf{T}}_t$.

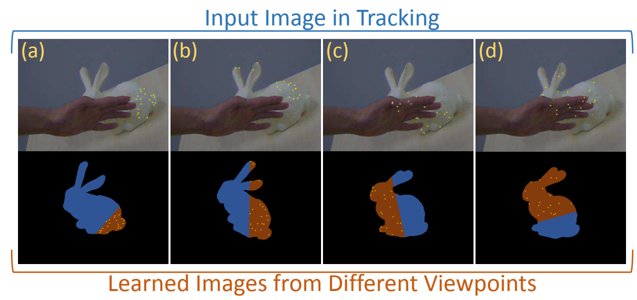


Figure 3. First row: occluded object when tracking. Second row: learned views where the occluded region is in blue and the points on the object, which are projected in the first row, are in yellow. Note that (a-b) are not affect by occlusion while (c-d) are affected.

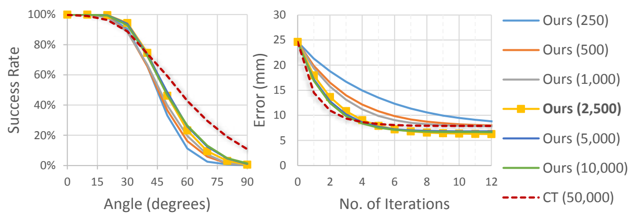
From the n_v views of the object, a subset of the trees are selected such that the object's viewpoint shows the highest similarity with the current frame. Using Eq. 3, $\hat{\mathbf{T}}_{t-1}$ generates the unit vector $\mathbf{N}_{t-1}$ that points to the camera in the object coordinate system. Then, the relation between the current view of the object from the learned views is measured through the angle between $\mathbf{N}_{t-1}$ and $\mathbf{N}_v$ for all views. Thus, the subset of trees chosen for evaluation is composed of the trees with the camera view that are within the neighborhood of $\mathbf{N}_{t-1}$, where the angle is less than θ.

To evaluate on the v-th view, $\boldsymbol{\epsilon}_{t-1}^v = [\epsilon_j^v(\hat{\mathbf{T}}_{t-1}; \mathbf{D}_t)]_{j=1}^{n_j}$ is constructed as the input to the trees. The threshold for $\boldsymbol{\epsilon}_{t-1}^v$ at each node guides the prediction to the left or right child until a leaf is reached. Each leaf stores the predicted mean and standard deviation of a parameter. After evaluating the trees from all neighboring views, the final prediction of a parameter is the average of the 20% predicted means with the least standard deviation. As a result, the average parameters are used to assemble the relative transformation $\mathbf{T}_t$ and we execute n_k iterations.

It is noteworthy to mention that, by taking the trees from a neighborhood of camera views and by aggregating only the best predictions, our algorithm can effectively handle large holes and occlusions. Indeed, as demonstrated in Fig. 3, some trees are affected by occlusion, the others can efficiently predict the correct parameters.

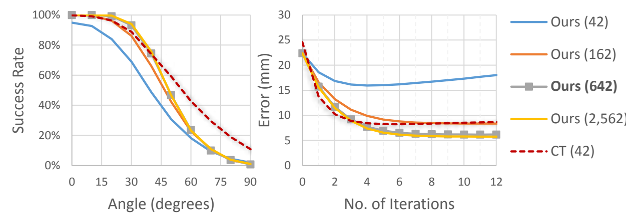
2.3. Online learning

When tracking an object in real scenes, there are situations when its 3D model is not at hand, which makes model-based tracking impossible. To track in these scenarios, we propose to deploy 3D *online* tracking, where, starting from a single 3D pose on an initial depth image, the target object is adaptively learned through the successive frames while being tracked, under unseen camera viewpoints and appearance changes. In contrast to learning a model-based tracker,



(a) Success Rate (b) Convergence Rate

Figure 4. (a) Success rate and (b) convergence rate of our proposal with varying sizes of the learning dataset compared against CT [23].



(a) Success Rate (b) Convergence Rate

Figure 5. (a) Success rate and (b) convergence rate of our proposal with different number of camera views in the geodesic grid compared against CT [23].

only the depth image $\mathbf{D}_v$ is given while the corresponding ground truth object transformation $\mathbf{T}_v$ is unknown.

For this approach, it is necessary to attain not only tracking efficiency but also learning efficiency. Our proposed tracking algorithm, through its attributes in terms of efficiency and memory footprint, suits nicely to this application. In particular, from one frame to the next, we propose to incrementally add new trees to the forest from different object viewpoint. To achieve this goal, the online learning is initialized by defining the object to learn in the first frame and a 3D bounding box that encloses the object. It follows that the centroid of the box is the origin of the object and the object transformation of the initial frame $\mathbf{T}_0$ is the translation from the camera center to the centroid. The bounding box defines the constraints of the object in 3D space and segments the object for learning. Thereafter, Sec. 2.1 is used to learn with the segmented image from $\mathbf{D}_t$ and the object transform $\hat{\mathbf{T}}_t$ as input. The initial frame needs to learn n_t trees per parameter to stabilize the forest for tracking the object in the next frames; while, the succeeding frames learn one tree per parameter. In this case, the geodesic grid from Fig. 2 is used to avoid re-learning trees from similar viewpoints. Thus, we find the closest vertex of the grid from the camera location in the object coordinate system and impose to have only one set of trees in each vertex.

3. Evaluation

This section evaluates the proposed tracking algorithm by taking into consideration, one at a time, the five essential attributes already discussed in Sec. 1 — (1) robustness, (2) tracking time and computational cost, (3) memory consumption, (4) scalability to multiple objects and (5) learning time. In addition to the results in this section, the qualitative evaluations from Fig. 1 are also reported in the form of videos in the *Supplementary Materials*.

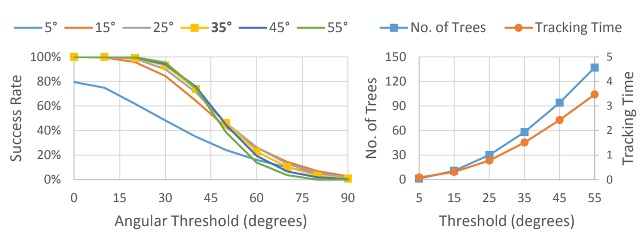
3.1. Robustness

To evaluate the robustness of our algorithm, we use three benchmark datasets [6, 11, 23]. The evaluation of the first dataset [11] determines the optimum parameters utilized throughout Sec. 3 and compares against the Chameleon Tracker (CT) [23]; the second [6] compares the accuracy of the transformation parameters against the RGB-D particle filter approaches [6, 13, 20]; finally, the third [23] compares the robustness of our approach against other trackers [1, 23] based on depth images only. Notably, across all the datasets, our work only uses the depth images of the RGB-D sequences.

Optimum Parameters. The driller dataset from [11] is composed of its model and 1,188 real RGB-D images with the ground truth pose of the object in each image. This evaluation focuses on the robustness of the algorithm to track an object in the current frame given its pose in the previous frame. To mimic the transformation of the previous frame, the ground truth pose is randomly translated and rotated using the Rodrigues' rotation formula [8, 18]. Thereafter, the tracker estimates the object's pose and the error of the estimated pose is computed based on the average distance between the corresponding vertices from the ground truth pose and the estimated pose.

From this error, the effects of different parameters on the tracker are observed through the success rate and the convergence rate. According to [11], a successfully estimated pose has the error value below 0.1 of the object's diameter. Moreover, the convergence rate takes the average error across the entire dataset for each of the iterations. These evaluations aim at finding the optimum parameters that produce the best results and to compare with CT [23].

In learning, there are two main aspects that affect the performance of the trees. These are the size of the learning dataset n_r and the number of camera views from the



(a) Success Rate (b) Tracking Time

Figure 6. (a) Success rate, and (b) tracking time and number of trees with respect to the angular distance threshold within the neighborhood of the camera location that is used in tracking.

	Errors	PCL	C&C	Krull	Ours	*Online*
(a) *Kinect Box*	t_x	43.99	1.84	**0.83**	1.54	2.25
	t_y	42.51	2.23	**1.67**	1.90	3.92
	t_z	55.89	1.36	0.79	**0.34**	1.82
	Roll	7.62	6.41	1.11	**0.42**	3.40
	Pitch	1.87	0.76	0.55	**0.22**	1.00
	Yaw	8.31	6.32	1.04	**0.68**	2.23
	Time	4539	166	143	**1.5**	1.1
(b) *Milk*	t_x	13.38	0.93	**0.51**	1.23	0.86
	t_y	31.45	1.94	1.27	**0.74**	1.02
	t_z	26.09	1.09	0.62	**0.24**	0.42
	Roll	59.37	3.83	2.19	**0.50**	1.66
	Pitch	19.58	1.41	1.44	**0.28**	1.14
	Yaw	75.03	3.26	1.90	**0.46**	1.29
	Time	2205	134	135	**1.5**	1.3
(c) *Orange Juice*	t_x	2.53	0.96	**0.52**	1.10	1.55
	t_y	2.20	1.44	**0.74**	0.94	1.64
	t_z	1.91	1.17	0.63	**0.18**	1.55
	Roll	85.81	1.32	1.28	**0.35**	2.94
	Pitch	42.12	0.75	1.08	**0.24**	2.37
	Yaw	46.37	1.39	1.20	**0.37**	4.71
	Time	1637	117	129	**1.5**	1.2
(d) *Tide*	t_x	1.46	0.83	**0.69**	0.73	0.88
	t_y	2.25	1.37	0.81	**0.56**	0.81
	t_z	0.92	1.20	0.81	**0.24**	0.36
	Roll	5.15	1.78	2.10	**0.31**	0.86
	Pitch	2.13	1.09	1.38	**0.25**	1.03
	Yaw	2.98	1.13	1.27	**0.34**	2.51
	Time	2762	111	116	**1.5**	1.2
Mean	Transl.	18.72	1.36	0.82	**0.81**	1.42
	Rot.	29.70	2.45	1.38	**0.37**	2.10
	Time	2786	132	131	**1.5**	1.2

Table 1. Errors in translation (mm) and rotation (degrees), and the runtime (ms) of the tracking results, evaluating with the synthetic dataset [6], of PCL [20], Choi and Christensen (C&C) [6], Krull *et al*. [13], and our approach with the model-based offline learning (Ours) as well as the image-based online learning (Online).

geodesic grid n_v. With regards to the size of the learning dataset, Fig. 4 illustrates that there is no significant difference in both success rate and convergence rate between 2500, 5000 and 10000; while, with the number of camera views, Fig. 5 shows that increasing from 642 to 2562 does not change the performance of the tracker. Thus, the optimum parameters for learning is 2,500 pairs of samples and labels with 642 camera views. Furthermore, based on the convergence rate in Fig. 4(b) and Fig. 5(b), 10 iterations ensures that the tracker converges to a low error value. We also look into the angular distance threshold θ of the neighboring trees when tracking. In Fig. 6(a), the success rate starts to converges with a threshold of 35°. On average, this corresponds to evaluate 58 trees from Fig. 6(b). For the rest of the evaluation, we use the parametric values of $n_r = 2500$, $n_v = 642$, $n_j = 20$ and $\theta = 35°$.

Compared to CT [23], we have a higher success rate when the relative motion is below 40°, while their success rate is higher above 40° in Fig. 4. Considering that an object temporal tracker estimates the transformation of the object between two consecutive frames, the success rates below 40° are, in terms of application, more relevant than the ones above. Furthermore, the error in their convergence rate initially drops faster than ours but we converge to a lower error value after as few as 4 iterations.

Synthetic Dataset. We evaluate our tracker on the publicly available synthetic dataset of [6]. It consists of four objects: each object has its model and 1,000 RGB-D images with the ground truth pose of the object. This evaluation aims at comparing the accuracy between the RGB-D particle filter approaches [6, 13, 20] and our method in estimating the rigid transformation parameters, *i.e*. the translation in the x-, y- and z-axis, and the roll, pitch and yaw angles.

Table 1 shows that we remarkably outperform PCL [20], and Choi and Christensen [6] over all sequences. With respect to [13], there is no significant difference in the error values: on average, we are 0.01 mm better in translation and 1.01° better in rotation. However, the difference between the two algorithms lies in the input data and the learning dataset. On one hand, they use RGB-D images while we only use the depth; on the other, their learning dataset includes the object's model on different backgrounds while we only learn using the object's model. The latter implies that they predefine their background in learning and limit their application to tracking objects in known (or similarly structured) environments. Due to this, their robustness in Table 1 depends on the learned background and, to achieve these error values, they need to know the object's environment beforehand. This is different from our work because we only use the object's model without any prior knowledge of the environment.

Real Dataset. This evaluation aims at comparing the robustness of the trackers that use depth images only, so to analyze in details the consequences in terms of tracking accuracy arising from the typical nuisances present in the 3D data acquired from consumer depth cameras.

To achieve our goal, we use the four real video sequences from [23] (see Fig. 7(a-d)) as well as an additional sequence with higher amount of occlusions and motion blur (see Fig. 7(e)). Each sequence is composed of 400 RGB-D images and the ground truth pose of the marker board. Across the frames of the sequence, we compute the average displacement of the model's vertices from the ground truth to the estimated pose. Moreover, we compare the robustness of ICP [1], CT [23] and our approach in the presence of large holes from the sensor, close-range occlusions from the surrounding objects and motion blur from the camera movement. We also compare our approach using sample points with random selection and with the selection to handle occlusions.

The first sequence in Fig. 7(a) is a simple sequence with small holes and small occlusions, where all trackers perform well. Next, the *driller* sequence illustrates the effects of large holes due to the reflectance of its metallic parts. It generates instability on CT [23] that is highlighted by the peak in Fig. 7(b). Even if it did not completely lose track of the object, this instability affects the robustness of the tracker in estimating the object's pose. On the contrary, ICP [1] and both versions of our method track the driller without any instability.

As reported in [23], the toy *cat* sequence in Fig. 7(c) causes ICP [1] to get trapped in a local minimum. When the cat continuously rotate until its tail is no longer visible due to holes and self-occlusion, the relatively large spherical shape of its head influences the error in the pose estimation and stays in that position for the succeeding frames. In contrast, CT [23] and our method track the toy cat without getting trapped in a local minimum.

The last two sequences in Fig. 7(d-e) present two important challenges. First, they exhibit close-range occlusions where the surrounding objects are right next to the object of interest. This introduces the problem in determining whether nearby objects are part of the object of interest or not. The second is the constant motion of both the camera and the object. This induces motion blur on the depth image which, in turn, distorts the 3D shape of the object.

Since the surrounding objects are close to the object of interest, ICP [1] fails in both sequences. When occlusions occur, it starts merging the point clouds from the nearby objects into the object of interest and completely fails tracking. For CT [23], it becomes unstable when occluded but completely recovers in Fig. 7(d). But, when larger occlusions are present such as Fig. 7(e), tracking fails. In comparison, our method with the random sample point arrangement has a similar robustness as [23]. However, our method, by modeling the sample points to handle occlusions smoothly, is able to track the object without any instability or failures. Among the competing methods, it is the only one that is able to successfully track the *bunny* in the last sequence.

3.2. Tracking time and computational cost

As witnessed in Fig. 6(b), the tracking time increases with respect to the number of trees evaluated or the number of camera views included. Using the optimum parameters from Sec. 3.1, the algorithm runs at 1.5 ms per frame on an Intel(R) Core(TM) i7 CPU, where only one core is used. This is comparable to the time reported by CT in [23]. With regards to the competing approaches [6, 13, 20] in Table 1,

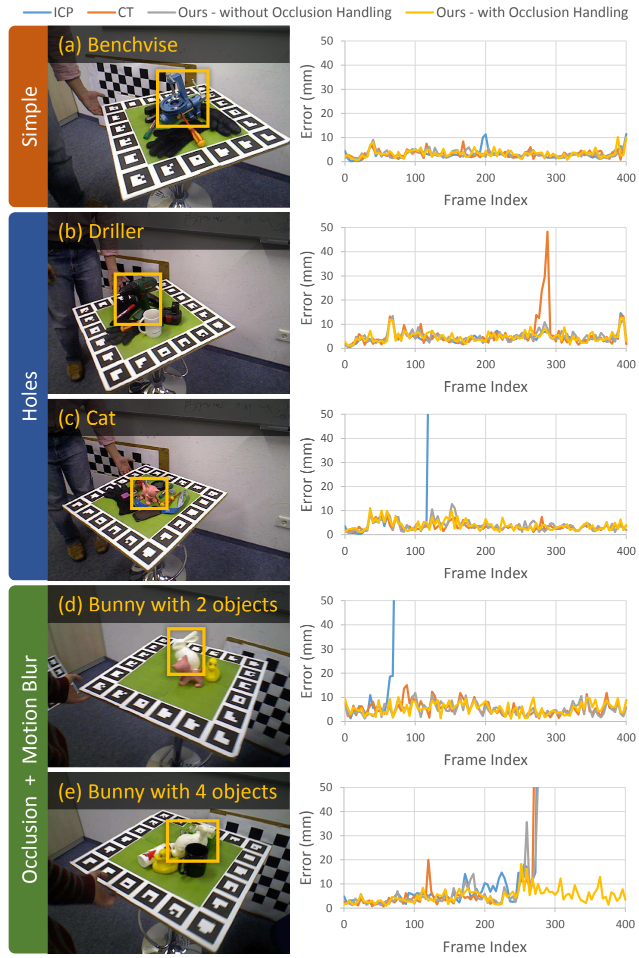


Figure 7. Tracking comparison on the dataset of [23] among ICP [1], CT [23], and our approach with and without the occlusion handling sample points selection.

their work takes about 100 times longer than ours while producing slightly higher error values. Among them, [6, 13] optimize their runtime through GPU.

3.3. Memory consumption

Our memory consumption increases linearly with the number of camera views and the size of the learning dataset for each view, as shown in Fig. 8(a) and (b), respectively. With the parameters from Sec. 3.1, our forests needs 7.4 MB. Compared to [23] which uses 821.3 MB, our memory requirement is two orders of magnitude less. Most of the related works do not mention or disregard this measurement from their papers, but we argue that it is an important aspect especially with regards to scalability towards tracking multiple objects.

3.4. Scalability to multiple objects

When tracking multiple objects, we utilize independent trackers for each object. It follows that the tracking time and

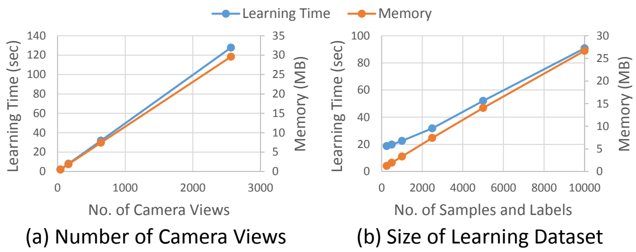


Figure 8. Learning time and memory usage with respect to (a) the number of camera views and (b) the size of the learning dataset.

the memory usage increase linearly with respect to the number of objects, where an increased computational expense, *i.e.* additional CPU cores, divides the resulting tracking time by the number of cores. Furthermore, the independence of the trackers for different objects keeps the robustness of the algorithm unaffected and the same as Sec. 3.1.

Considering a typical computer with 8 GB RAM and 8 CPU cores, a memory consumption of 7.4 MB for each object allows us to include more than 1,000 objects into RAM. In contrast to CT [23] where they use 821.3 MB for each object and reached a maximum limit of 9 objects, our tracker can include at least two orders of magnitude more objects in memory than [23].

To demonstrate the remarkable scalability of our approach, we synthetically rendered 108 moving objects with random initial rotations in a 3D video sequence as shown in Fig. 1(a); apply one tracker for each object, requiring a total memory footprint of, approximately, 108×7.4 MB $= 799.2$ MB; and, track them independently. By using 8 CPU cores, our work tracks all 108 objects at 33.7 ms per frame, *i.e.* yielding a frame rate of 30 fps. Interestingly, our memory requirement for 108 objects is less than that required for just one object by CT; while, our tracking time for 108 objects is less than the GPU implementations of [6, 13] that track one object at 130 ms per frame. It is important to mention that we had to resort to a rendered input video given the difficulty of recreating a similar scenario with so many moving objects under real conditions. Nevertheless, since we only aim at evaluating the scalability of our approach, we can expect an identical performance under real conditions.

Therefore, although scalability is linear with respect to the number of objects, we highlight the extremely low magnitude of all the important components — (2) tracking time and computational cost, and (3) memory consumption — that makes tracking a hundred objects at real-time possible.

3.5. Learning time

The learning time has a linear relation with respect to the number of camera views and the size of the learning dataset as shown in Fig. 8. Thus, with the optimum parameters from Sec. 3.1 and 8 CPU cores, it requires 31.8 seconds to learn the trees from all of the 642 camera views using 2,500 pairs of sample and label. This is significantly lower than the 12.3 hours of CT [23]. Even with an increased number of camera views or a larger learning dataset in Fig. 8, our learning time remains below 140 seconds.

Online learning. One of the most interesting outcomes of the fast learning time is the online learning where the tracker does not require the object's 3D model as input to learning.

We use the dataset of [6] to evaluate our online learning strategy. In the first frame, we use the ground truth transformation to locate the object and start learning with 50 trees per parameter, which takes 1.3 seconds. The succeeding frames continues to learn one tree per parameter, which takes 25.6 ms per frame. In Table 1, the average tracking error of the online learning is comparable to the results of Choi and Christensen [6]. It performs worse than the model-based trackers with offline learning of Krull *et al.* [13] and ours, but performs better than PCL [20]. Furthermore, the combined learning and tracking time, which is approximately 26.8 ms per frame using 8 CPU cores, is still faster than the competing approaches [6, 13, 20] that only execute tracking. Notably, a simple practical application of the online learning, where the object's model is not at hand, is the head pose estimation in Fig. 1(c).

3.6. Failure cases

Since we are tracking the geometric structure of the objects through depth images, the limitation of the tracker is highly related to its structure. Similar to ICP [3, 5], highly symmetric objects loses some degrees of freedom in relation to its axis of symmetry. For instance, a bowl that has a hemispherical structure loses one degree of freedom because a rotation around its axis of symmetry is ambiguous when viewed from the depth image. Therefore, although our algorithm can still track the bowl, it fails to estimate the full 3D pose with six degrees of freedom.

Specific to online learning, large holes or occlusions in the initial frames create problems where the forest has not learned enough trees to describe the object's structure. Due to this, drifts occur and tracking failures are more probable.

4. Conclusions and future work

We propose a real-time, scalable and robust 3D tracking algorithm, that can be employed both in model-based as well as in online 3D tracking. Throughout the experimental evaluation, our approach demonstrated to be flexible and versatile so to adapt to a variety of 3D tracking applications, contrary to most 3D temporal tracking approaches that are designed for a fixed application. Analogously to what was done in [14] between 2D tracking and Structure-from-Motion, an interesting future direction is to deploy the versatility of our approach for "bridging the gap" between 3D tracking, 3D SLAM and 3D reconstruction.

References

[1] Documentation - point cloud library (pcl). http://pointclouds.org/documentation/tutorials/iterative_closest_point.php. 5, 6, 7

[2] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. Di Stefano, and M. Vincze. Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In *International Conference on Robotics and Automation*, pages 2104–2111. IEEE, 2013. 2

[3] P. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992. 2, 8

[4] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 2, 3

[5] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992. 2, 8

[6] C. Choi and H. I. Christensen. Rgb-d object tracking: A particle filter approach on gpu. In *International Conference on Intelligent Robots and Systems*, pages 1084–1091. IEEE, 2013. 2, 5, 6, 7, 8

[7] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 998–1005, June 2010. 2

[8] L. Euler. Problema algebraicum ob affectiones prorsus singulares memorabile. *Commentatio 407 Indicis Enestoemiani, Novi Comm. Acad. Sci. Petropolitanae*, 15:75–106, 1770. 5

[9] A. W. Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13):1145–1153, 2003. 2

[10] R. Held, A. Gupta, B. Curless, and M. Agrawala. 3d puppetry: A kinect-based interface for 3d animation. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, 2012. 2

[11] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision*, pages 548–562. Springer, 2013. 2, 5

[12] C.-H. Huang, E. Boyer, N. Navab, and S. Ilic. Human shape and pose tracking using keyframes. In *Conference on Computer Vision and Pattern Recognition*, pages 3446–3453. IEEE, 2014. 2

[13] A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, and C. Rother. 6-dof model based tracking via object coordinate regression. 2, 5, 6, 7, 8

[14] K. Lebeda, S. Hadfield, and R. Bowden. 2d or not 2d: Bridging the gap between tracking and structure from motion. In *Asian Conference on Computer Vision*, 2014. 8

[15] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE, 2011. 2

[16] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013. 2

[17] C. Y. Ren, V. Prisacariu, O. Kaehler, I. Reid, and D. Murray. 3d tracking of multiple objects with identical appearance using rgb-d input. In *International Conference on 3D Vision*, volume 1, pages 47–54. IEEE, 2014. 2

[18] O. Rodrigues. Des lois géometriques qui regissent les déplacements d' un systéme solide dans l' espace, et de la variation des coordonnées provenant de ces déplacement considerées indépendent des causes qui peuvent les produire. *J. Math. Pures Appl.*, 5:380–400, 1840. 5

[19] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling*, 2001. 2

[20] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, pages 1–4. IEEE, 2011. 5, 6, 7, 8

[21] R. Sadourny, A. Arakawa, and Y. Mintz. Integration of the nondivergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere 1. *Monthly Weather Review*, 96(6):351–356, 1968. 3

[22] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Robotics: Science and Systems*, 2009. 2

[23] D. J. Tan and S. Ilic. Multi-forest tracker: A chameleon in tracking. In *Conference on Computer Vision and Pattern Recognition*, pages 1202–1209. IEEE, 2014. 2, 3, 5, 6, 7, 8