Multi-Forest Tracker: A Chameleon in Tracking

David Joseph Tan
Technische Universität München
tanda@in.tum.de

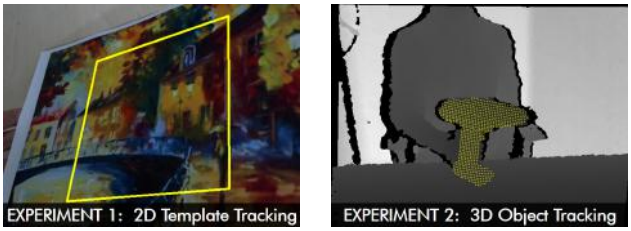
Slobodan Ilic
Technische Universität München
slobodan.ilic@in.tum.de

Abstract

In this paper, we address the problem of object tracking in intensity images and depth data. We propose a generic framework that can be used either for tracking 2D templates in intensity images or for tracking 3D objects in depth images. To overcome problems like partial occlusions, strong illumination changes and motion blur, that notoriously make energy minimization-based tracking methods get trapped in a local minimum, we propose a learning-based method that is robust to all these problems. We use random forests to learn the relation between the parameters that defines the object's motion, and the changes they induce on the image intensities or the point cloud of the template. It follows that, to track the template when it moves, we use the changes on the image intensities or point cloud to predict the parameters of this motion. Our algorithm has an extremely fast tracking performance running at less than 2 ms per frame, and is robust to partial occlusions. Moreover, it demonstrates robustness to strong illumination changes when tracking templates using intensity images, and robustness in tracking 3D objects from arbitrary viewpoints even in the presence of motion blur that causes missing or erroneous data in depth images. Extensive experimental evaluation and comparison to the related approaches strongly demonstrates the benefits of our method.

1. Introduction

Object tracking either in 2D or 3D can be formulated using an energy minimization where it finds a set of parameters by minimizing an objective function. It has widely been used through the work of Lucas and Kanade [15] for 2D template tracking and through the ICP algorithm [5] for 3D object tracking using point clouds. Both methods use a set of initial parameters and iteratively refine them by minimizing an objective function to reach a set of optimum parameters. Thus, the choice of an optimizer and parametrization is crucial to avoid local minima and to converge quickly to the optimum solution. However, factors, such as lighting changes, partial occlusions and fast motions, typically lead the energy-based minimization to fail and get trapped at a local minimum.



(a) Intensity Image (b) Depth Image

Figure 1. These images show tracking results from our two experiments where we used our theoretical work to implement (a) a 2D template tracking algorithm with intensity images as the input medium; and, (b) a 3D object tracking algorithm with depth images as the input medium.

Contrary to this, learning-based approaches learn the relation between the parameters that define the relative movement of a template and the data of the template described by this motion. Henceforth, when the template moves, the change of the data on the template can predict the parameters of this movement. In this way, they are extremely fast and can jump over local minima. One of the prominent works in this field is from Jurie and Dhome [13] where they use linear regression to learn the relation between the changes in warping parameters and the changes in image intensities on the template. Their work also shows the superiority of learning in contrast to its energy minimization counterpart.

Theoretically, we can modify the linear regression to track 3D templates on depth images by replacing the objective function that involves 3D point clouds. Unfortunately, in practice, it does not work because the depth data from the sensor is unstable. For example, if the depth camera is filming a static environment, we observe a constant change in the depth values due to the amount of noise and missing data especially at the object discontinuities. These values are different from the learned template and can be interpreted as small occlusions that are spread over the template. Therefore, the algorithm for 3D object tracking requires the

capacity to handle partial occlusions. However, linear regression is highly sensitive to occlusions as reported in [14].

Inspired by this problem, we designed a learning-based tracking algorithm using random forests that handles partial occlusions, but retains good properties of linear regression [13] such as speed and robustness. The reason for choosing random forest is because it is an ensemble of trees where each tree learns and predicts independently from the other. Ideally, when some trees get affected by occlusion, the others can still generate good predictions.

Our main contribution is a generalized learning-based tracking framework, which we describe as a *chameleon* because it can adapt to different input modalities. In addition, we also contribute by demonstrating applications of this method to 2D template tracking in Sec. 4.1 and to full 3D object tracking from any viewpoint in Sec. 4.2. The strong attributes of both experiments includes robustness to partial occlusion and remarkable speed of less than 2 ms per frame. Furthermore, the 2D tracker is robust to strong illumination changes while the 3D tracker is robust to noise from fast motion.

2. Related work

Frame-to-frame tracking can be divided into two categories that are based on energy minimization and learning. The main difference between them is that the former is generally slower and is more sensitive to local minima; whereas, the latter requires an extensive training procedure. To have a more focused comparison of our approach with other methods in these categories, the scope of this section is limited to model-based frame-to-frame tracking using intensity (or RGB) images, depth images or both.

Energy minimization-based approach. The work of Lucas and Kanade [15] has significantly triggered an advancement in the field of 2D template tracking. Baker and Matthews [3] has summarized these through four different update rules – additive approach [15], compositional approach [20], inverse additive approach [6, 8], and inverse compositional approach [2, 7]. Among them, the inverse additive and inverse compositional approaches have decreased tracking time by switching the roles of the source and target images to evaluate several computations prior to tracking.

On the other hand, ICP [5] and variations of it [18, 19] has dominated the research field regarding 3D object tracking. However, ICP has problems when foreign objects such as clutter or hands are close to or occlude the object of interest. For instance, when tracking hand-held objects in [9], the authors segments the hand through the intensity image to remove the point clouds associated to it before running ICP. Another approach in 3D object tracking includes the level sets of Ren *et al*. [17] that uses a probabilistic method to statistically determine occlusions. But this also uses intensity images as an appearance model to help handle occlusions.

Learning-based approach. Using the objective function of an energy-minimization approach, Jurie and Dhome [13] builds up from the work of Hager and Belhumeur [8] to learn linear regression to find the relation of the intensities and parameters by randomly warping the 2D template using different parameters of the warping function. Then, when the template is moved, the changes in intensities can predict the warping parameters. Another work is from Mayol and Murray [16] where they use general regression to fit the sampling region to pre-trained samples.

There have been successful attempts to handle occlusion using 2D templates. In [11, 14, 21], they represent a template using smaller templates so that, when the template is partially occluded, only a few smaller templates are affected. However, in addition to occlusions from other objects where a specific region of the template is affected, noise and missing data from the sensor can be interpreted as small occlusions, appearing as small curves or small blobs, that extend throughout the template. As a result, this could affect a significant number of small templates used in [11, 14, 21] to handle occlusions. Furthermore, unlike textured images, tracking small templates is unstable in 3D because, if we divide the surface of an object into small portions, most of them have a very similar structure to their neighbors which makes tracking ambiguous.

To the best of our knowledge, we have implemented the first learning-based tracking algorithm that uses depth images alone. Nevertheless, our algorithm is a generic approach that is applicable to both intensity and depth images. Furthermore, it is robust to occlusions, illumination changes as well as fast motion, and runs in less than 2 ms per frame.

3. Method

Our work is rooted from the objective function of Hager and Belhumeur [8] where they relate the image intensities of a template and transformation parameters by the pseudo-inverse of a Jacobian matrix. However, in contrast to [8], our method uses random forests in lieu of the Jacobian matrix, which generalizes it to any input function and not constrained to 2D intensity images.

3.1. Objective function

Given an arbitrary input function Ω_t at time t (*e.g.* intensity images or point clouds), the location of the reference template is represented by n_s sample points $\{\mathbf{x}_s \in \mathbb{R}^N\}_{s=1}^{n_s}$ at t_0 such that the template is described by the set of values $\{\Omega_{t_0}(\mathbf{x}_s), \forall\, \mathbf{x}_s\}$. As the template moves across time, the sample points are transformed as $\Phi(\boldsymbol{\mu}) \circ \mathbf{x}_s$, where the vector $\boldsymbol{\mu}$ contains the parameters of the transformation function

Φ and its initial value is $\boldsymbol{\mu}_{t_0} = \mathbf{0}$. Therefore, at time t, the objective function minimizes:

$$\varepsilon(\boldsymbol{\mu}_t) = \sum_s \left|\Omega_t(\Phi(\boldsymbol{\mu}_t) \circ \mathbf{x}_s) - \Omega_{t_0}(\mathbf{x}_s)\right|^2 \quad (1)$$

such that, at time $t+\tau$, the parameter vector $\boldsymbol{\mu}_t$ updates to $\boldsymbol{\mu}_{t+\tau} = \boldsymbol{\mu}_t + \delta\boldsymbol{\mu}$ by minimizing:

$$\varepsilon(\delta\boldsymbol{\mu}) = \sum_s \left|\Omega_{t+\tau}(\Phi(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}) \circ \mathbf{x}_s) - \Omega_{t_0}(\mathbf{x}_s)\right|^2 \quad (2)$$

where $\delta\boldsymbol{\mu}$ is the parameter update vector. To simplify this equation, we assign $\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = [\Omega_t(\Phi(\boldsymbol{\mu}) \circ \mathbf{x}_s)]_{s=1}^{n_s}$ as a collection of $\Omega_t(\cdot)$; hence, Eq. 2 is rewritten in vector form as:

$$\varepsilon(\delta\boldsymbol{\mu}) = \left\|\boldsymbol{\Omega}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}, t+\tau) - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0)\right\|^2. \quad (3)$$

Similar to [3, 8, 13], Eq. 3 can be formulated as:

$$\delta\boldsymbol{\mu} = -\mathbf{J}^+_{\boldsymbol{\mu}_t}[\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0)] \quad (4a)$$

$$= -\mathbf{J}^+_{\boldsymbol{\mu}_t}\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau) \quad (4b)$$

where the $\mathbf{J}_{\boldsymbol{\mu}}$ is the Jacobian matrix of $\boldsymbol{\Omega}$ with respect to $\boldsymbol{\mu}$, $\mathbf{J}^+_{\boldsymbol{\mu}} = (\mathbf{J}^\top_{\boldsymbol{\mu}}\mathbf{J}_{\boldsymbol{\mu}})^{-1}\mathbf{J}^\top_{\boldsymbol{\mu}}$, and $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = \boldsymbol{\Omega}(\boldsymbol{\mu}, t) - \boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0)$. Therefore, the pseudo-inverse of the Jacobian matrix $-\mathbf{J}^+_{\boldsymbol{\mu}_t}$ in Eq. 4 represents the relation from the given $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau)$ to the parameter update $\delta\boldsymbol{\mu}$. In this way, $\delta\boldsymbol{\mu}$ updates the transformation function as:

$$\Phi(\boldsymbol{\mu}_{t+\tau}) = \Phi(\boldsymbol{\mu}_t) \circ \Phi(\boldsymbol{\mu}_{t_0} + \delta\boldsymbol{\mu}) = \Phi(\boldsymbol{\mu}_t) \circ \Phi(\delta\boldsymbol{\mu}). \quad (5)$$

Instead of finding the non-linear relation $-\mathbf{J}^+_{\boldsymbol{\mu}_t}$ in Eq. 4, we formulate a learning method using random forests to find the relation between $\delta\boldsymbol{\Omega}$ and $\delta\boldsymbol{\mu}$.

3.2. Tracking with random forests

We use regression forests to learn how different values of $\delta\boldsymbol{\mu}$ affect the template in Ω_{t_0} through $\delta\boldsymbol{\Omega}$. Subsequently, when the input function $\Omega_{t+\tau}$ and the parameters $\boldsymbol{\mu}_t$ are given, the forests use $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau)$ to predict $\delta\boldsymbol{\mu}$ and update the transform to $\Phi(\boldsymbol{\mu}_{t+\tau})$. Contrary to [8], this learning scheme is not restricted to 2D images and can handle different types of input function Ω without the necessity or difficulty to derive different Jacobian matrices.

It is noteworthy to mention that our method assumes independence between the parameters in $\boldsymbol{\mu}$ and learns one forest for each parameter separately. This results to n_p forests where n_p is the number of parameters in $\boldsymbol{\mu}$ and each forest consists of n_t trees.

Training forests. This process begins by creating a training dataset where we produce n_ω random values of $\delta\boldsymbol{\mu}^\omega$ to transform the input Ω_{t_0} to Ω^ω, such that the location of the sample points in Ω^ω is computed as $\Phi(\delta\boldsymbol{\mu}^\omega) \circ \mathbf{x}_s$. It follows that we can define the set $\mathcal{S} = \{(\delta\boldsymbol{\Omega}^\omega, \delta\boldsymbol{\mu}^\omega_p)\,\forall\,\omega\}$ that is used to construct the p-th forest where the vector $\delta\boldsymbol{\Omega}^\omega = \delta\boldsymbol{\Omega}(\mu_{t_0}, \omega)$ is the input sample of the forest and the scalar $\delta\boldsymbol{\mu}^\omega_p$ is the p-th parameter in $\delta\boldsymbol{\mu}^\omega$. In general, the objective of using such a synthetic training dataset is to generate $\mathcal{S}$; thus, there are several ways of creating $\mathcal{S}$ and using a synthetic dataset by transforming Ω_{t_0} is just one of them.

Before training a tree, we randomly select n_r points from the n_s sample points of the template and only use these points to construct the tree. The goal is to impose randomness and to help handle cases when some sample points are not available or have incorrect values in Ω (*e.g.* occlusion). Hence, we assign the features $\{\theta_r\}_{r=1}^{n_r}$ as the indices of the θ_r-th sample point.

Each feature is used to split the samples that arrive in the node $\mathcal{S}_N$ into two subsets $\mathcal{S}_l$ and $\mathcal{S}_r$ that goes to its left and right child, respectively. These subsets are defined as:

$$\mathcal{S}_l = \left\{(\delta\boldsymbol{\Omega}^\omega, \delta\boldsymbol{\mu}^\omega_p) \in \mathcal{S}_N \,|\, \delta\boldsymbol{\Omega}^\omega_{\theta_r} \geq \kappa_{\theta_r}\right\} \quad (6a)$$

$$\mathcal{S}_r = \left\{(\delta\boldsymbol{\Omega}^\omega, \delta\boldsymbol{\mu}^\omega_p) \in \mathcal{S}_N \,|\, \delta\boldsymbol{\Omega}^\omega_{\theta_r} < \kappa_{\theta_r}\right\} \quad (6b)$$

where $\delta\boldsymbol{\Omega}^\omega_{\theta_r}$ is the θ_r-th element of $\delta\boldsymbol{\Omega}^\omega$, and κ_{θ_r} is the threshold. Furthermore, to evaluate the split, we use the information gain to determine whether it produced less random or more homogeneous subsets, that is written as:

$$I(\theta_r) = \sigma(\mathcal{S}_N) - \sum_{i\in\{l,r\}} \frac{|\mathcal{S}_i|}{|\mathcal{S}_N|}\sigma(\mathcal{S}_i) \quad (7)$$

where $\sigma(\mathcal{S}_i)$ is the standard deviation of all $\delta\boldsymbol{\mu}^\omega_p$ in $\mathcal{S}_i$. Among all θ_r, we look for the best feature with the highest information gain. Consequently, the node stores the best feature and its threshold, and passes the subsets to its children. Note that the choice of the threshold κ_{θ_r} depends on the values of all $\delta\boldsymbol{\Omega}^\omega_{\theta_r}$ in $\mathcal{S}_N$. It can either be a single threshold such as the median, or multiple threshold candidates such as linearly spaced values. Moreover, if multiple thresholds are used, each of them is also evaluated using Eq. 7 and the one with the highest information gain is stored.

The tree continuously splits the samples and grows until at least one of these stopping criteria is true: (1) the maximum depth of the tree is reached; (2) the number of samples $|\mathcal{S}_N|$ is small; (3) $\sigma(\mathcal{S}_N)$ is sufficiently low; or, (4) the information gain of the best feature is less than a threshold. In this case, the node is considered as a leaf and stores the mean and standard deviation of all $\delta\boldsymbol{\mu}^\omega_p$ that reached this leaf. Similarly, the same process occurs for all n_t trees in each forest and for all n_p forests.

Predicting $\delta\boldsymbol{\mu}$ in tracking. Using the parameter vector $\boldsymbol{\mu}_t$ on the current input function $\Omega_{t+\tau}$, we compute the input sample $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_t, t+\tau)$ of the the forests to predict $\delta\boldsymbol{\mu}$

that updates the parameters. Starting from the root node of each tree, the input sample uses the splitting parameters in the node to determine whether to go to the left or right child, and continuously traverse from the parent node to the child node until it reaches a leaf where the learned mean and standard deviation that predict a parameter of $\delta\boldsymbol{\mu}$ are stored. As a result, each of the n_p parameters have n_t predictions.

To find the final prediction of a parameter in $\delta\boldsymbol{\mu}$, we take a percentage of the n_t predictions with the lowest learned standard deviation and aggregate them by taking the average of the learned means. Finally, we update the transform by employing Eq. 5 and also update the location of the sample points in $\Omega_{t+\tau}$. Lastly, we iteratively repeat the entire process to refine the previously predicted parameters.

4. Experiment

This section focuses on two exemplary experiments that utilized our approach with different input function Ω. The first experiment involves tracking templates in 2D intensity images; while, the second tracks 3D objects in depth images. Interestingly, both experiments are robust in tracking, can handle partial occlusion, and track in less than 2 ms using one core of the CPU.

4.1. 2D template tracking

Our method is applied as a 2D template tracking algorithm using intensity images Ω. Without loss of generality, we parameterize a rectangular template using the 2D location of its four corners $\{\mathbf{x}_c\}_{c=1}^4$ at t_0, and define its motion by the displacement of these corners $\{\delta\mathbf{x}_c\}_{c=1}^4$. From these, the transformation function $\Phi(\boldsymbol{\mu})$ is denoted as a homography that transforms $\{\mathbf{x}_c\}_{c=1}^4$ to $\{\mathbf{x}_c+\delta\mathbf{x}_c\}_{c=1}^4$ where $\boldsymbol{\mu} = [\delta\mathbf{x}_c]_{c=1}^4$and $\Phi(\boldsymbol{\mu}_{t_0}) = \mathbf{I}_{3\times3}$. Moreover, we position the sample points $\{\mathbf{x}_s\}_{s=1}^{n_s}$ on an $n_s = n_g \times n_g$ regular grid fitted into the template.

Considering the vector $\boldsymbol{\Omega}$ from Sec. 3, it follows that $\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = [\Omega_t(\Phi(\boldsymbol{\mu}) \cdot \mathbf{x}_s)]_{s=1}^{n_s}$ is a collection of image intensities on the image Ω_t at the transformed sample points, and $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, \omega) = [\Omega^\omega(\mathbf{x}_s) - \Omega_{t_0}(\mathbf{x}_s)]_{s=0}^{n_s}$ is the input sample of the forest where $\Omega^\omega(\mathbf{x}_s)$ is the warped template and $\Omega_{t_0}(\mathbf{x}_s)$ is the reference template. Hence, in training, the synthetic data are transformations of the image Ω_{t_0} with n_ω random motions from $\delta\boldsymbol{\mu}$. Moreover, after predicting the parameters $\delta\boldsymbol{\mu}$ in tracking, we update the homography as $\Phi(\boldsymbol{\mu}_{t+\tau}) = \Phi(\boldsymbol{\mu}_t) \cdot \Phi(\delta\boldsymbol{\mu})$.

To handle illumination changes, we normalize the n_r sample points in each tree such that the intensities on these points have zero mean and unit standard deviation.

Parametrization. Using this concept, we use a 250×250 template and train $n_t = 50$ trees for each $n_p = 8$ forests with 25×25 grid enclosed on the template and $n_s = 625$ sample points. For the synthetic dataset, we generate $n_\omega = 50,000$ transformed images where the corners moves in the range of $[-85, 85]$ pixels. The thresholding in each branch uses 10 linearly spaced values and the stopping criteria in each node includes maximum depth of 20, minimum number of samples of 40, minimum standard deviation of 0.5, and minimum information gain of 0.01. In tracking, we aggregate 20% of the prediction with the lowest standard deviation and run 15 iterations.

Evaluation. Using the same dataset and evaluation as in [12], we measure the tracking robustness of our algorithm and compare it with Jurie and Dhome [13] where they learn the relation between image intensity difference and parameter update vector using linear regression. The choice in comparing with [13] is due to its similarity with our approach where both are rooted from [8] and replace the pseudo-inverse of the Jacobian matrix with a learning algorithm. Moreover, it is reported in [12] that the overall performance of [13] is either comparable to or better than [4, 11, 12].

Here, we learn the template at the center of the image and track this template after warping the image. The tracking robustness is computed as the percent of successfully tracked templates where the average distance of the tracked corners to its ground truth location is less than 5 pixels when backwarped. In addition, we compare our tracking robustness with different values of n_r and, based on the plots in Fig. 2, the performance converges when $n_r = 25$. From these evaluations, the average tracking time of our approach is approximately 1.47 ms when using 1 core of the CPU, while linear regression [13] runs at 0.47 ms. Thus, their approach is 1 ms faster than ours which is almost negligible.

With different types of image warping in Fig 2(a-d), we show that our 2D experiment has similar tracking robustness as the linear regression [13]. This robustness is reinforced in Fig. 3(a) where we demonstrate qualitative examples of real-time tracking. For the evaluation with respect to noise in Fig. 2(e) where we randomly translate the image by a maximum distance of 35 pixels after inducing varying levels of Gaussian noise, it illustrates that we are slightly better in tracking than [13].

However, our learning approach is much more robust to occlusion than [13] as shown in Fig. 2(f) where the image is translated after replacing a percent of the template's region by a different image; while, in Fig. 3(c), we demonstrate its performance after occluding the template with different objects. Note that this property becomes a requirement when some of the pixel values from the sensor is not available such as shadows from depth sensors – a thorough investigation in using our algorithm for depth images is conducted in Sec. 4.2. In addition, we also demonstrate in Fig. 3(b) that we are also robust to strong illumination changes which [13]

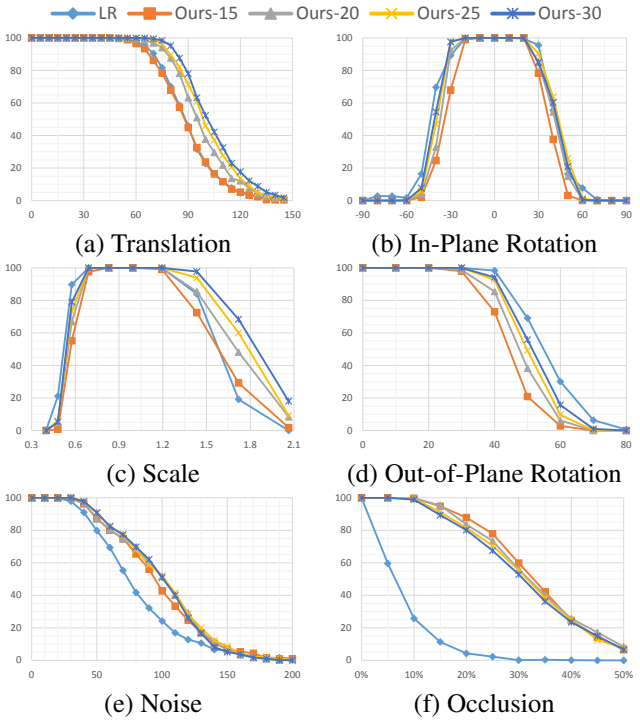


Figure 2. These plots show the tracking robustness of our algorithm with varying n_r (15, 20, 25, 30) and compare it with Linear Regression (LR) [13] under different (a-d) transformations, (e) levels of Gaussian noise and (f) percentage of occluded region.

Figure 3. These images illustrate the essential capabilities of our 2D template tracking algorithm – (a) tracking with perspective transform, (b) tracking under strong illumination changes, and (c) tracking with partial occlusion. More examples are demonstrated in the *Supplementary Materials*.

cannot handle.

Therefore, although there are similarities with [13], our algorithm has proven to be a *chameleon* since its range of application is not limited to a specific sensor and its tracking robustness is not compromised by this generality. With regards to 2D tracking, we have demonstrated that both algorithms are equally robust in tracking performance, but we are more robust in the presence of occlusion and strong illumination changes.

4.2. 3D object tracking

We designed a model-based tracking method that finds the pose of a 3D rigid object using the depth image $\mathbf{D}_t$. Here, we are using the object coordinate system, where the centroid of the model's vertices is the origin, and the camera coordinate system, where the camera center is the origin. The camera is parameterized by a 3×3 intrinsic matrix $\mathbf{K}$ and a 4×4 extrinsic matrix $\mathbf{E}$ that relates the camera coordinate system and the object coordinate system at t_0. Moreover, the sample points $\{\mathbf{X}_s\}_{s=1}^{n_s}$ are 3D homogeneous points on the model as seen by the camera coordinate system which implies that the corresponding points in the object coordinate system are computed as $\{\mathbf{E}^{-1}\mathbf{X}_s\}_{s=1}^{n_s}$.

Looking from the object coordinate system, the rigid transform of the object is defined by the 4×4 matrix:

$$\mathbf{T}_{obj}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma) \quad (8)$$

where $\mathbf{t} = (t_x, t_y, t_z)^\top$ is the translation vector; α, β and γ are the yaw, pitch and roll angles, respectively; and, the parameter vector $\boldsymbol{\mu}$ is composed of the 3 translation parameters and 3 rotation parameters. Therefore, after transforming the object, the sample points in the camera coordinate system transforms as:

$$\mathbb{P}_{\boldsymbol{\mu}}(\mathbf{X}_s) = \mathbf{E}\mathbf{T}_{obj}(\boldsymbol{\mu})\mathbf{E}^{-1}\mathbf{X}_s \quad (9)$$

where $\mathbb{P}_{\boldsymbol{\mu}_{t_0}}(\mathbf{X}_s) = \mathbf{X}_s$; then, the projection of the points into the camera's image is given as $\mathbf{x}_s^d = [\mathbf{K}|\mathbf{0}] \cdot \mathbb{P}_{\boldsymbol{\mu}}(\mathbf{X}_s)$ where $\mathbf{x}_s^d$ is given in 2D homogeneous coordinates.

If we denote $\mathbb{D}_t(\mathbf{x})$ as the backprojection of the pixel $\mathbf{x}$ in the depth image $\mathbf{D}_t$, then, to find the optimum $\delta\boldsymbol{\mu}$ at $t+\tau$, we minimize the sum of the distances:

$$\left\| \mathbb{D}_{t+\tau}([\mathbf{K}|\mathbf{0}]\mathbb{P}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbf{X}_s)) - \mathbb{P}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbf{X}_s) \right\|^2 \quad (10)$$

$$= \left\| \mathbb{P}^{-1}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbb{D}_{t+\tau}([\mathbf{K}|\mathbf{0}]\mathbb{P}_{\boldsymbol{\mu}_t+\delta\boldsymbol{\mu}}(\mathbf{X}_s))) - \mathbf{X}_s \right\|^2 \quad (11)$$

for all sample points. However, we observed that the difference in the x- and y-coordinates are close to 0 in Eq. 11; hence, we simplify the error function by only using the difference in the z-coordinates:

$$\varepsilon(\delta\boldsymbol{\mu}) = \sum_s |\Phi_{t+\tau}(\boldsymbol{\mu}_t + \delta\boldsymbol{\mu}) \circ \mathbf{X}_s - \mathbf{X}_s|_z^2 \quad (12)$$

where $\Phi_t(\boldsymbol{\mu}) \circ \mathbf{X}_s = \mathbb{P}_{\boldsymbol{\mu}}^{-1}(\mathbb{D}_t([\mathbf{K}|\mathbf{0}]\mathbb{P}_{\boldsymbol{\mu}}(\mathbf{X}_s)))$ is a time-varying transformation function that is dependent on the depth image $\mathbf{D}_t$, and the operator $|\cdot|_z$ takes the z-coordinate of the point. When we compare Eq. 12 with Eq. 3, the vector $\boldsymbol{\Omega}(\boldsymbol{\mu}, t) = [|\Phi_t(\boldsymbol{\mu}_t) \circ \mathbf{X}_s|_z]_{s=1}^{n_s}$ is described as a collection of the z-coordinates of the transformed sample points where $\boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, t_0) = [|\mathbf{X}_s|_z]_{s=1}^{n_s}$.

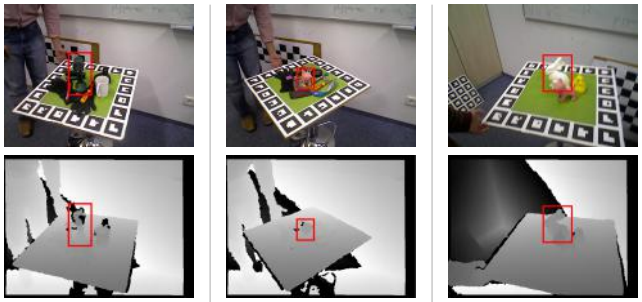
In training, the input sample can be further simplified as $\delta\boldsymbol{\Omega}(\boldsymbol{\mu}_{t_0}, \omega) = [|\mathbb{D}^{\omega}([\mathbf{K}|\mathbf{0}]\mathbf{X}_s) - \mathbf{X}_s|_z]_{s=1}^{n_s}$, which changes with respect to the depth image across ω. As a consequence, to create the set $\mathcal{S}$ for training, we render n_ω depth images with different parameters of $\delta\boldsymbol{\mu}^{\omega}$ in $\mathbf{T}_{obj}$. Moreover, from the synthetic depth image with $\mathbf{T}_{obj}(\boldsymbol{\mu}_0)$, the model on the image is enclosed by an $n_g \times n_g$ regular grid, where the points on the model are backprojected and are used as the sample points. On the other hand, in tracking, the prediction $\delta\boldsymbol{\mu}$ updates $\mathbf{T}_{obj}$ in the transformation function Φ as $\mathbf{T}_{obj}(\boldsymbol{\mu}_{t+\tau}) = \mathbf{T}_{obj}(\boldsymbol{\mu}_t) \cdot \mathbf{T}_{obj}(\delta\boldsymbol{\mu})$.

Multi-view tracking. The problem with our sample point arrangement is that it is restricted to one view of the object. For instance, if the object keeps rotating in one direction, at some point, the tracker becomes unstable and fails since the number of visible sample points continuously decreases. Due to this, we individually learn random forests for n_c views of the object where the camera is located around the object using an icosahedron. It follows that the c-th camera view has its own set of sample points $\{\mathbf{X}_s^c\}_{s=1}^{n_s^c}$ and extrinsic matrix $\mathbf{E}_c$, and this produces $6 \cdot n_c \cdot n_t$ trees. Therefore, using the object coordinate system in tracking, we find the closest learned camera view to the current position of the camera. Mathematically, to switch from one view to the other, we modify Eq. 9 to:

$$\mathbb{P}_{\boldsymbol{\mu}}^{c}(\mathbf{X}_s^c) = \mathbf{E}_0 \mathbf{T}_{obj}(\boldsymbol{\mu}) \mathbf{E}_c^{-1} \mathbf{X}_s^c \tag{13}$$

where $\mathbf{E}_0$ is the initial extrinsic matrix in tracking.

Parametrization. Based on this, we use $n_t = 100$ trees for each $n_p = 6$ parameters with a 40×40 grid enclosed on the model as seen from the depth image where only points that lie on the model are used as sample points. In training, we render $n_\omega = 50,000$ depth images where the object randomly transforms with a translation that ranges in $[-35, 35]$ mm for each axis, and the angles α, β and γ in $[-30°, 30°]$. Moreover, each tree randomly chooses $n_r = 20$ sample points, each branch uses 10 linearly spaced thresholds and each node checks the stopping criteria with depth of 20, 40 samples, standard deviation of 0.5, and information gain of 0.01. This is done for $n_c = 42$ camera views of the model. Then, in tracking, we aggregate 15% of the predictions with the lowest standard deviation and run 10 iterations.



(a) Driller (b) Cat (c) Bunny

Figure 4. These images show the frames (a) when our approach becomes unstable in the driller sequence due to the lack of depth data, and (b-c) when ICP fails in the cat and bunny sequences. Note that the object of interest is marked in red.

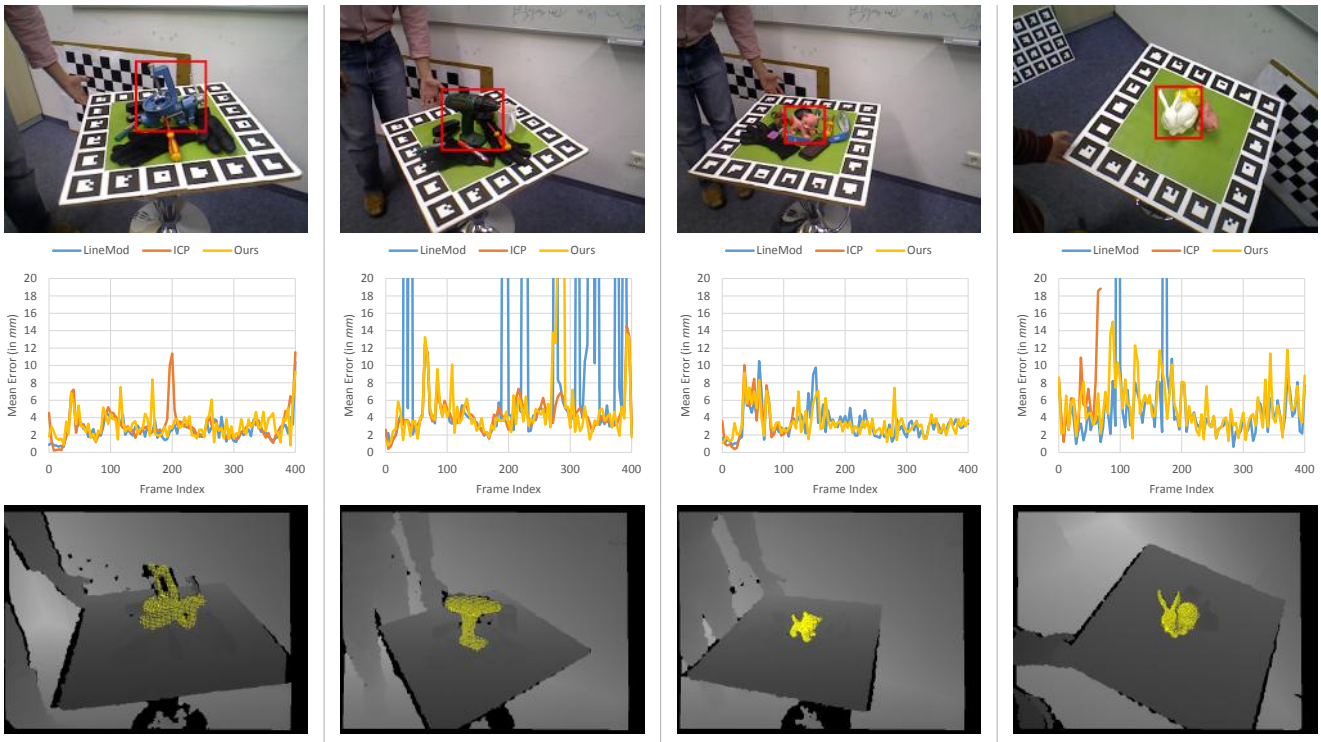
Evaluation. For this experiment, we created a dataset with four sequences using PrimeSense's Carmine 1.09 camera to track different objects – vise, driller, cat and bunny – in cluttered environments as shown in the first row of Fig. 5. We compare our results to the ICP implementation from PCL [1] where the vertices of the model are used as input source, and LineMod [10], which is the state-of-the-art 3D object detector, that is taken from the original implementation of the authors. Note that LineMod refines the pose of the object after template matching using ICP.

To generate ground truth transforms, we placed the objects on top of a board with markers on its edges; thus, we can transform the object's model using the ground truth and compare it with the location of the tracked object through the mean distance of the model's vertices as plotted in the second row of Fig. 5. Moreover, in the vise, driller and cat sequences, the camera stayed static while the board rotates; but, in the bunny sequence, both the camera and board moves. Hence, from this dataset, the tracking results can be summarized as follows:

1. **VISE**. All approaches works well in this sequence even if there is a small occlusion from the red screwdriver as illustrated in Fig. 5(a).

2. **DRILLER**. For this sequence, there are some depth images that does not have sufficient information for a portion of the object as shown in Fig. 4(a). This affects both LineMod and our method. However, it is important to mention that our method became unstable in a small section of the sequence and it did not lose tracking; while, LineMod frequently fails in detecting the object.

3. **CAT**. As the cat rotates, the depth image loses information of its tail as shown in Fig. 4(b) and its relatively large spherical head dominates the ICP algorithm. Due to its shape, ICP stayed static and fails to track the rotation of the cat. This results in a tracking failure for the



(a) Vise (b) Driller (c) Cat (d) Bunny

Figure 5. The first row shows the setup of the four sequences that are used to evaluate the 3D tracking algorithm where the object of interest is mark in red; the second shows the mean distance error for each sequence using LineMod [10] where their peaks indicate detection failures, PCL's ICP [1] where ICP fails at frame 116 in (c) and frame 72 in (d), and our approach; and, the last row shows our tracking results in the corresponding depth image.

rest of the images in the sequence. Regarding LineMod and our algorithm, they work well in this sequence and smoothly track the cat's rotation without getting trapped in a local minimum.

4. **BUNNY**. There are two essential criteria to consider in this sequence. The first is the fast motion of both object and camera which creates a motion blur in the colored image as well as noise in the depth image as shown in Fig. 4(c), and the second is the closeness of the surrounding objects to the object of interest as depicted in Fig. 5(d). In this sequence, the bunny is occluded by the cat, then the duck. Therefore, when the cat occluded a large portion of the bunny, ICP started incorporating the cat as part of the bunny and loses tracking. For LineMod, the detector also fails when the objects occlude the bunny, which is indicated by the two peaks in Fig. 5(d). Finally, our approach became unstable but completely recovers after the occlusion.

On average, our tracking time is 1.75 ms for the vise, 1.76 ms for the driller, 1.84 ms for the cat and 1.84 ms for the bunny when using only 1 core of the CPU. For ICP, the average tracking time is 189.09 ms for the vise, 72.24 ms for the driller, 65.04 ms for the cat and 225.72 ms for the bunny. Moreover, the average tracking time for LineMod is approximately 119 ms for all models.

Furthermore, we illustrate some tracking examples in Fig. 6 from a video where the actor plays with the cat, picks it up through its tail, turns it around, then drops it (see *Supplementary Materials*). These examples show how well the tracker handles occlusion. For instance, when the cat is occluded by the hand in Fig. 6(a), it is not necessary to do any pre-processing procedure, such as segmenting the hand, even if the object is relatively small. Additionally, when the cat is turned or dropped, it also shows how well our algorithm handles fast motion.

Taking the results from the dataset into account, we have exhibited different scenarios and show that, in comparison to LineMod and ICP, our 3D algorithm can avoid local minima, is more stable in the presence of both noise and occlusion; and, is at least 35 times faster than ICP [1] and two orders of magnitude faster than LineMod [10]. Through the examples in Fig. 6, we demonstrate that we have a good tracking performance in fast motion as well as partial occlusion.

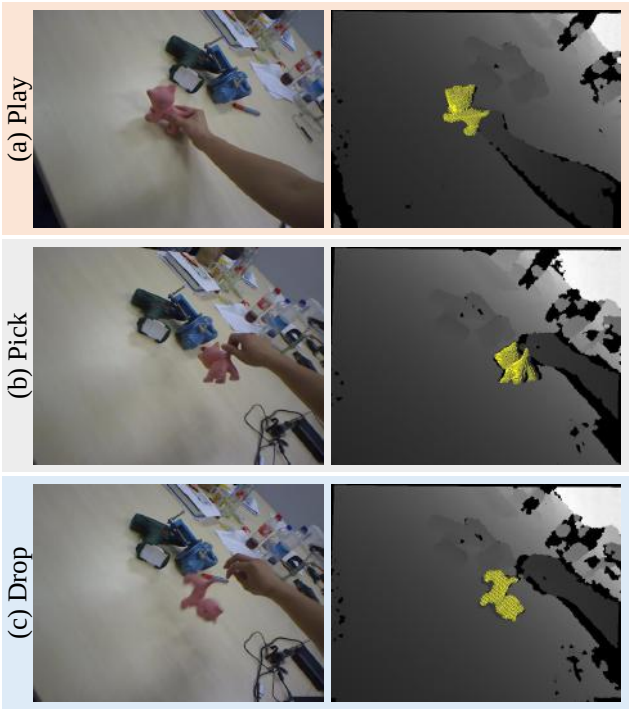


Figure 6. These images show examples of our 3D tracking algorithm where the actor (a) plays with the cat; (b) picks it up; then, (c) drops it. More examples are in the *Supplementary Materials*.

5. Conclusion

This paper introduces a generic tracking algorithm that we describe as *chameleon* because it can be adapted to different input modalities. To show the value of our work, we designed two experiments where the first tracks 2D templates using intensity images while the other tracks 3D objects using depth images. Although both experiments are robust in tracking and track in less than 2 ms when using one core of the CPU, individually, they have merits of their own. The 2D template tracking algorithm can handle partial occlusions and strong illumination changes. Moreover, after comparing our approach with ICP [1] and LineMod [10], we can conclusively say that our 3D tracker outperforms the two approaches in the presence of partial occlusion at close proximity and noise due to fast motion.

References

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

[2] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *Conference on Computer Vision and Pattern Recognition*, 2001.

[3] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 2004.

[4] S. Benhimane and E. Malis. Homography-based 2d visual tracking and servoing. *International Journal of Robotics Research*, 2007.

[5] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992.

[6] M. Cascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[7] F. Dellaert and R. Collins. Fast image-based tracking by selective pixel integration. In *ICCV Workshop of Frame-Rate Vision*, 1999.

[8] G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.

[9] 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.

[10] 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*, 2012.

[11] S. Holzer, S. Ilic, and N. Navab. Adaptive linear predictors for real-time tracking. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[12] S. Holzer, S. Ilic, D. Tan, and N. Navab. Efficient learning of linear predictors using dimensionality reduction. In *Asian Conference on Computer Vision*, 2012.

[13] F. Jurie and M. Dhome. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[14] F. Jurie and M. Dhome. Real time robust template matching. In *British Machine Vision Conference*, 2002.

[15] B. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conference on Artificial Intelligence*, 1981.

[16] W. W. Mayol and D. W. Murray. Tracking with general regression. *Machine Vision and Applications*, 2008.

[17] C. Ren, V. Prisacariu, D. Murray, and I. Reid. Star3d: Simultaneous tracking and reconstruction of 3d objects using rgb-d data. In *International Conference on Computer Vision*. 2013.

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

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

[20] H.-Y. Shum and R. Szeliski. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, 2000.

[21] K. Zimmermann, J. Matas, and T. Svoboda. Tracking by an optimal sequence of linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.