Multimedia Systems (2023) 29:401–420
https://doi.org/10.1007/s00530-022-00996-6

REGULAR PAPER

DATaR: Depth Augmented Target Redetection using Kernelized Correlation Filter

Srishti Yadav[1] · **Shahram Payandeh**[1]

Received: 6 January 2022 / Accepted: 17 August 2022 / Published online: 6 October 2022

Abstract
Unlike deep learning which requires large training datasets, correlation filter-based trackers like Kernelized Correlation Filter (KCF) use implicit properties of tracked images (circulant structure) for training in real time. Despite their popularity in tracking applications, there exists significant drawbacks of the tracker in cases like occlusions and out-of-view scenarios. This paper attempts to address some of these drawbacks with a novel RGB-D Kernel Correlation tracker in target re-detection. Our target re-detection framework not only re-detects the target in challenging scenarios but also intelligently adapts to avoid any boundary issues. Our results are experimentally evaluated using (a) standard dataset and (b) real time using the Microsoft Kinect V2 sensor. We believe this work will set the basis for improvement in the effectiveness of kernel-based correlation filter trackers and will further the development of a more robust tracker.

Keywords Visual tracking · Depth-based tracking · Correlation filters · Kinect sensors

1 Introduction

Visual tracking as a field has seen tremendous progress in recent years in robotics and surveillance applications. It is trying to address the issues caused by noise, clutter, occlusion, illumination changes, and viewpoints (e.g., in mobile or aerial robotics). There have been numerous attempts in designing and deploying a robust tracking method. However, achieving full tracking accuracy under realistic conditions presents various challenging scenarios. Despite the widespread applications of neural network-based architectures, recent years have also seen a significant shift in the attention towards trackers that learn "on the fly" i.e., approaches that model how an object varies visually over time, as and when new data becomes available. Many discriminative Correlation Filter (CF) tracking methods have adopted this approach. Though neural network-based architectures (or various deep learning extensions) have shown good accuracy, they have significant disadvantages in terms of cost, training time, and computational power: For example, XLNet model [1] costs $61,000 to train, uses 512 TPU v3 chips with a batch size of 2048 (for comparison, a person or a small research lab normally uses a batch size of 32 with normal compute) and takes 2.5 days to train. These issues have encouraged the visual tracking community to look for a faster and more competitive alternative to CF-based trackers. They offer solutions for real-time tracking with good real-time performance. Correlation filters have gotten significant attention because of their high frame per second (FPS), low computation power requirement (they work significantly well with CPUs but can be made faster with few GPUs), and high efficiency. One such tracker is the Kernelized Correlation Filter (KCF) [2] tracker, which is a type of correlation filter-based tracker known for its ability in handling thousands of sample data yet keep the computation load low by exploring tools of kernel trick and properties of Circulant Matrices. Despite the progress made in tracking robustness, most works like [3–5] mostly focus on scale adaptation, occlusion detection, or shape change. A big part of long-term tracking is not only detection of occlusion but also re-detection of the target once it is out of the occlusion and tracking It in real-time continuously.

Communicated by R. Huang.

✉ Srishti Yadav
srishtiy@sfu.ca

Shahram Payandeh
payandeh@sfu.ca

[1] Networked Robotics and Sensing Laboratory, School of Engineering Science, Simon Fraser University, Burnaby, Canada

Our work specifically focuses on human targets that are occluded by objects (e.g., chairs) or by other humans (e.g. a tall person). It discusses the implementation of architecture that can help overcome this challenge. The tracking methodology proposed considers a single Kinect RGB-D camera, single-target, and is model-free that is applied to long-term tracking. The model-free property means that the only supervised training example is provided by the bounding box in the first frame. Long-term tracking means that the tracker learns to re-detection after the target is lost. i.e., to infer the object's position in the current frame. This paper proposes a novel architecture to enable the KCF tracker to be more robust during occlusion by utilizing additional depth information. A key advantage of the proposed tracker is that the depth information used by the tracker is intelligently adapted to avoid boundary issues (situations where a target may be at the edge of the camera view or about to leave the camera view). For initialization, our tracker, similar to KCF, uses the Region of Interest (ROI) specified in the first frame. However, KCF only uses image features of the RGB image within the ROI but our proposed tracker uses both image features and depth information within the ROI. Due to the inclusion of depth information, this approach can lead to boundary issues since the depth data is expected to change at the edge when the tracker is in motion, adversely affecting the tracking performance. However, as it will be shown, the depth information in the proposed tracker is intelligently adapted to incorporate the correct depth information of the moving tracker. Our proposed tracker was validated on the Princeton RGB-D dataset as well as the real-time dataset collected by the author. The results show that the tracker can successfully detect when the human target is occluded and re-detect it after occlusion.

This paper is divided into five sections. The paper first gives a literature review of the RGB and RGB-Depth-based trackers in Sect. 3 followed by the details of the algorithm in Sect. 4. We discuss our experimental set-up, including the description of the dataset in Sect. 5 which is followed by results and observations in Sect. 6. We conclude the paper in Sect. 7 where we discuss the findings of our proposed long-term RGB-D tracker.

2 Literature review on RGB and RGB-D based trackers

Since the invent of MOSSE [6] in 2010, correlation filter (CF) based trackers gained huge popularity owing to their speed and accuracy. Correlation Filters are a class of classifiers, which use a designed template to produce sharp peaks (strong correlation) in the correlation output. The peaks correspond to the precise localization of the object/target in scenes. One of the popular CF-based trackers was proposed by Henrique et al. where the authors proposed a kernelized version of the CF tracker [7] which benefitted from the circulant structure of the samples. This tracker was studied in [8–10] and further improvements were investigated by [11, 12] who applied a correlation filter to scale space, addressing the issue of scale adaptation. Other improvements include spatial regularization in SRDCF [13], continuous convolution in C-COT [14], max-margin classifiers in [15], Spatio-temporal learning [16], and adding robustness using part-based features [17, 18].

Overall, several efforts have been made to address where challenging issues of visual tracking. However, target appearance, if used as the main cue for tracking, is not a very reliable feature when the target suffers from challenging issues of occlusion, out-of-view, and illumination changes. Features like depth data, with its ability to distinguish between foreground and background, can help in making the tracker more accurate. Trackers have been developed in the past which use RGB features augmented with additional features like depth. Reference [19] build upon color-only KCF tracker and adds depth showing a real-time performance of ~ 35 FPS. Reference [20] proposes a distractor-aware learning method (DLS) with RGB-D data to effectively alleviate the model drift problem. Reference [21] uses depth features with RGB to address color camouflage issue.

These recent improvements in RGB-based CF-based trackers have come at a cost of speed and real-time performance. For example, the Discriminative Correlation Filter (DCF) [13] using HOG features reached ~ 6 FPS as compared to some early state-of-arts like KCF [7] which attained ~ 170 FPS, and MOSSE [6] which was ~ 700 FPS. Similarly adding depth has helped improve accuracy but the challenging issues are yet to be resolved. It would be interesting to explore an improved CF tracker which can address some of the existing issues related to robustness e.g., occlusion, model drift, scale changes, color camouflage, etc., yet achieves higher accuracy. Few works have attempted to address this gap of speed in RGB-D trackers. Reference [4] proposes a deep depth-aware long-term tracker that extends a deep discriminative correlation tracker (DCF) to embed depth information to deep features. It achieves state-of-the-art RGB-D tracking performance and has better speed performance. Closing the gap between speed and accuracy in an RGB-D tracker is an ongoing problem.

3 Depth augmented target re-detection

KCF is based on the discriminative method, which formulates the tracking problem as a binary classification task and distinguishes the target from the background using a discriminative classifier [22]. Two of the main factors of the efficiency of KCF are that it uses augmentation of negative samples to

enhance the discriminative ability while also exploring the structure of the circulant matrix and performing the computation in the Fourier domain for high speed. The training and detection pipeline of KCF can be seen in (Fig. 1).

KCF algorithms work as follows: Given the initial selection of a target (i.e., center position and size), a tracked region is created. The tracked region is increased from the target size to provide some context. This tracked region is a part of an image frame. Various features are extracted from the tracked region and each channel is weighted by a cosine window. A circulant matrix is used to learn all the possible shifts of the target from a base sample. The coefficient α_f encodes (Eq. 1) the training samples, consisting of all shifts of a base sample in the frequency domain. It is these shifts that provide a large number of samples of the training. The fast-learning equation is expressed as:

$$\alpha_f = \frac{y_f}{k_f^{xx'} + \lambda}, \tag{1}$$

where $y_f = F_y$ denotes the Discrete Fourier Transform (DFT) of y. The term $k_f^{xx'}$ denotes the DFT of $k^{xx'}$; the kernel correlation function between signals x and x'. The division represents an element-wise division and the scalar λ is a regularization term. The training label matrix y is a Gaussian function that smoothly decays from the value of one for the centered target to zero for other shifts.

The response is computed as the element-wise multiplication between the learnt $\widetilde{\alpha}_f$ and the correlation of z_f with the learnt model $\widetilde{x}_f$ at various image patches z. The detection response for each location is:

$$r\left(k_f^{z\widetilde{x}}\right) = F^{-1}\left(k_f^{z\widetilde{x}} \odot \widetilde{\alpha}_f\right), \tag{2}$$

where $r\left(k_f^{z\widetilde{x}}\right)$ is the response, $\widetilde{x}_f$ and alpha $\widetilde{\alpha}_f$ are the linear interpolations of x_f and α_f at each detection with the selection of an interpolation factor and $\odot$ represents the dot product. The readers are directed to [7] for more details. Hence, we can increase the computation speed due to: (a) element-wise multiplication in Fourier Domain and (b) computation in high dimensional data using kernel trick. Once we are able to locate the target, these target locations can be used to interpolate our model to train the data in a new frame (at this location). This method can be employed for all subsequent frames. Hence, for any instance in the image, with the past target location and current features, the target can be tracked over time. This type of tracking methodology is called *tracking-by-detection* and has been used in the research time and again.

3.1 Method implementation

As discussed previously, KCF tracker achieves faster throughput by replacing convolution actions in the spatial domain with element-wise multiplication in the frequency domain using RGB color features. However, these color features do not encode complete tracking information about the target. Depth information can provide additional spatial information in the form of distance to the sensor. This additional information forms the basis of our occlusion detection and re-detection framework. This is based on the assumption that an occluding object will have a different depth of information as compared to the target being occluded.

The target occlusion knowledge is computed using the information from around the target's center. It is so because the information at the edges of the bounding box tends to include small false information, but the target center is

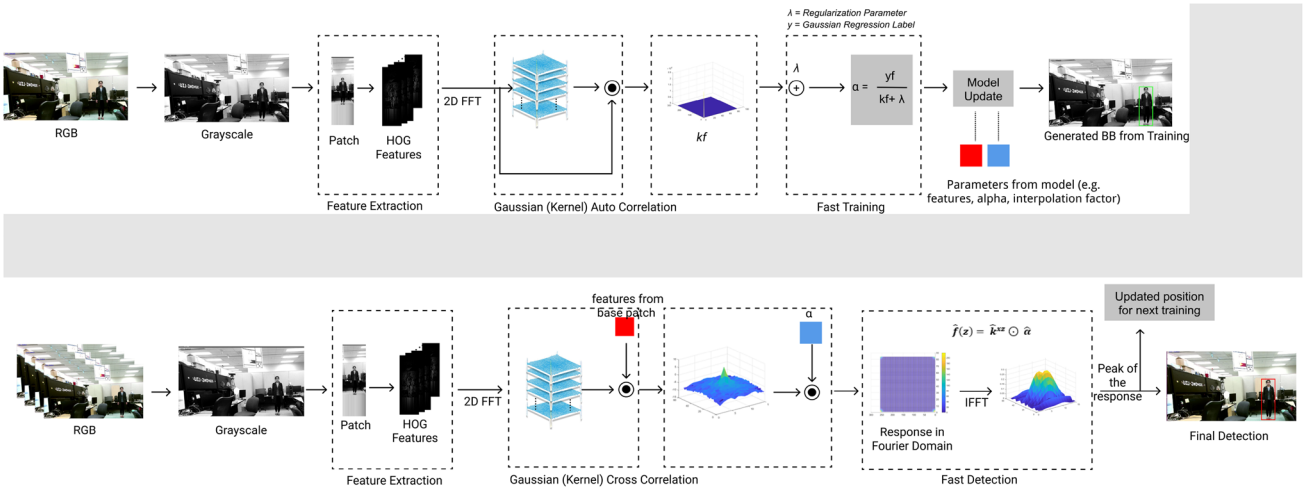


Fig. 1 Training and detection pipeline of KCF

expected to be constant. Since the computation is made on a small section (the area around the target center) of the target bounding box area, it improves computational efficiency. The RGB information helps the tracker to update the model template (updated in the Fourier domain) and depth information helps it to decide when to track. Our proposed tracker stops tracking at the moment the target is occluded. Hence, it adds to the tracker's robustness as the model and its coefficients are interpolated correctly. In any tracking algorithm involving depth data, ensuring that the correct depth of the target is included is very important. KCF with depth data takes the RGB image features and depth information of the target patch using the bounding box used for detection. As the target moves and changes its position or scale, there is a high probability that the bounding box (of the target) starts including more depth data of the background from the edges. This would falsely provide higher background depth, as compared to (only) target depth, making it harder for the model to make the right predictions.

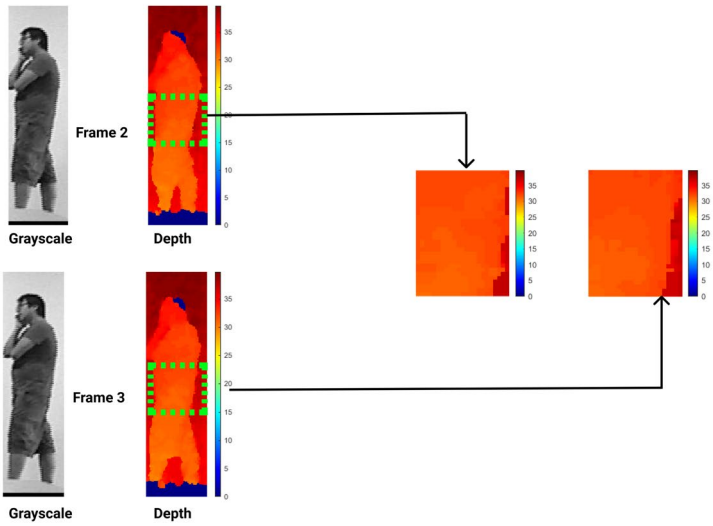
Our tracker proposes a self-adjusting depth patch as a solution to this problem. It saves the depth information of the previous (target) patch and new (target) patch and locates the possible center of these depth patches by taking the area around center of the patch (calculated using position coordinates). Figure 2a and b show the difference between a frame and a patch. Figure 2c shows how depth patch around the center of two patches is extracted.

3.2 Creating self-adjusting depth patch

The tracker further computes the difference in the depth information of these two parts of patches and looks for any high peaks at the edge of this value as shown in Fig. 3. The edges are defined as a few columns on the extreme end. For a non-occluding object, any abnormally high peak at the edges would indicate the inclusion of background depth information (which is undesired). This is based on the assumption that two depth images in two subsequent frames will have

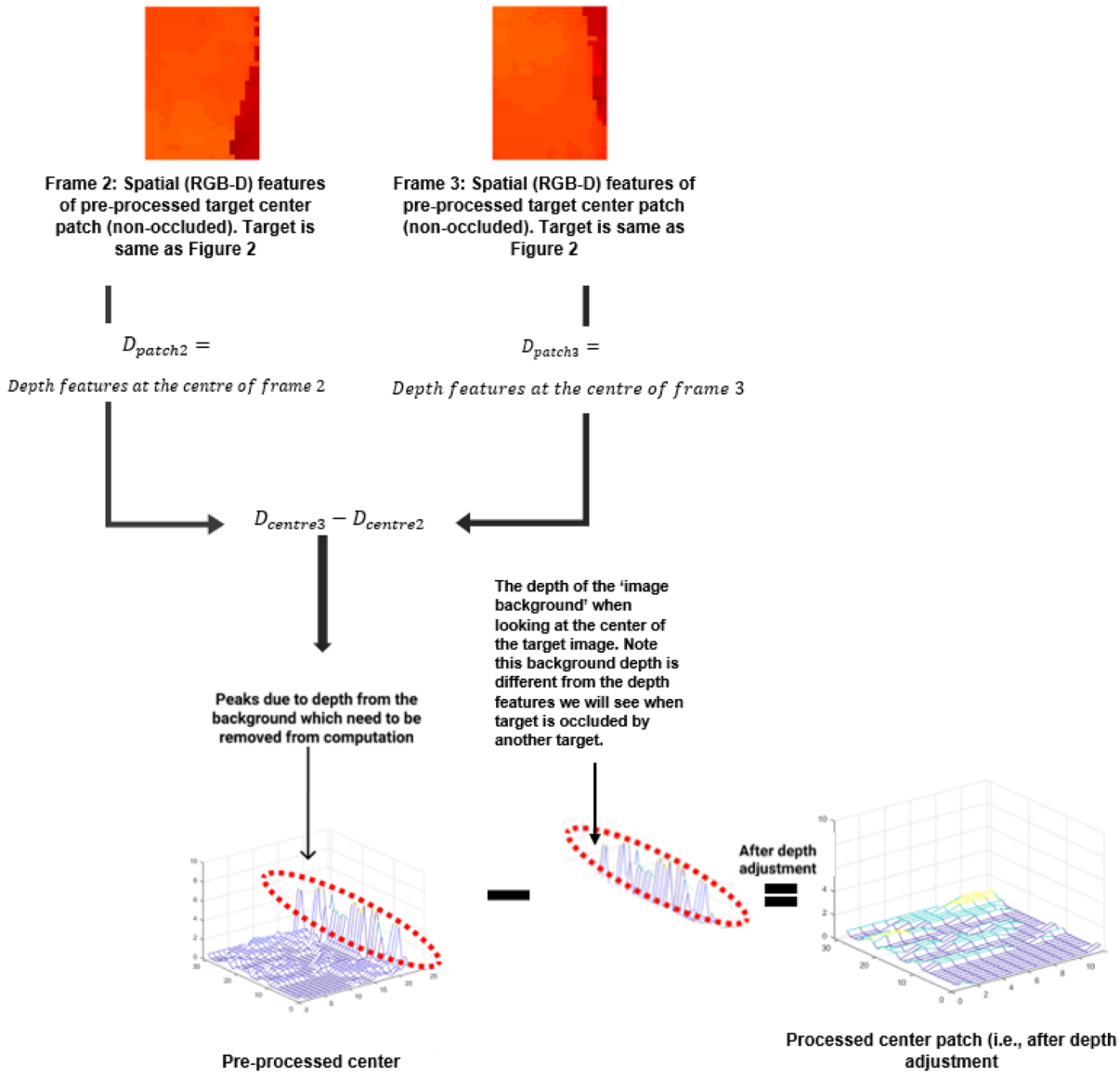
(a) Frame (grayscale)

(b) Patch from the frame



(c) Depth patch around the center of two patches of two subsequent frames

Fig. 2 Figure shows **a** grayscale frame, **b** patch from the grayscale frame, **c** the depth of the patch around the center of two patches (of two subsequent frames)



(Bottom Left) Before depth adjustment: Mesh graph of difference of centre depth patch for frame 2 and frame 3 i.e., $D_{centre3} - D_{centre2}$ where x = width, y = height, z = depth value

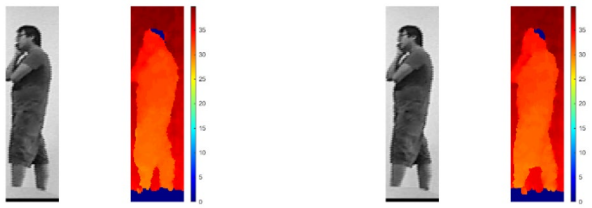
(Bottom Right) After depth adjustment: Mesh graph of difference of centre depth patch for frame 2 and frame 3; x = width, y = height, z = depth value

Fig. 3 The figure shows how the difference in center depth patch is computed for two subsequent frames when the target is not occluded. The image frame and the patch considered here are the same as in Fig. 2

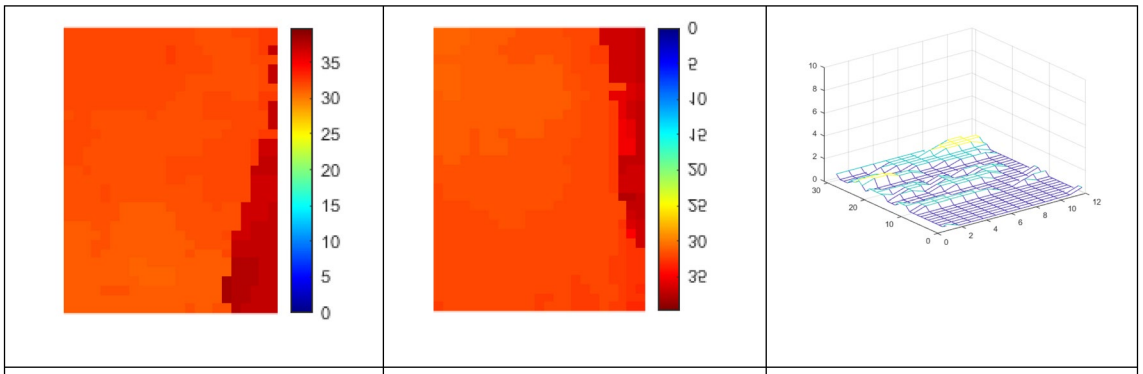
similar depth data around their center, hence their difference should be minimal or close to zero.

If there is a high peak, we decrease the size of the depth patches from the edges such that abnormally high peaks are removed as shown in Fig. 4. After the depth has been adjusted, the difference in the depth is minimal ensuring that we have correct depth information for the target.

However, in the case of occlusion, even after adjusting this difference in depth patch of subsequent frames, peaks will exist because the peaks will cover far more than the area at extreme edges (with few peaks almost towards the center of the patch) as shown in Fig. 5. The existence of these peaks helps the tracker identify the occurrence of occlusion in the frame.



Frame 2: Grayscale and Depth Frame 3: Grayscale and Depth



(a)	(b)	(c)
Centre depth patch: Depth of the area around target patch center) extracted from the depth data of the target patch of frame 2	Centre depth patch: Depth of the area around target patch center) extracted from the depth data of the target patch of frame 3	(After depth adjustment) Difference for center depth patches of two (non-occluding) frames (Frame2 and Frame 3). We see no peaks in their difference after depth adjustment.

Fig. 4 The figure shows how the difference in center depth patch changes for two subsequent frames when the target is not occluded

3.3 Training, detection, and re-detection of target under occlusion

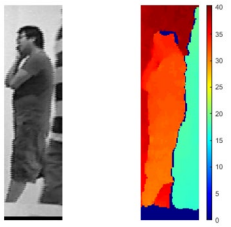
Our proposed RGB-D tracking with a re-detection algorithm builds upon the KCF tracker. KCF uses image data features to locate and detect trackers. In our proposed tracker, we add depth information to provide the tracker with contextual information about the target and the background, for it to be able to distinguish between the two. The tracking pipeline for the proposed RGB-D tracker is shown in Fig. 6, the symbols of which are defined as follows:

σ	Standard deviation
λ	Regularizer
α	Learnable parameter
$f(z)$	The response of kernel ridge regression
$\text{patch}_{\text{rgb}}$	Part of the RGB image which has the target, as shown in Fig. 2. b
pos_i	Position of the target defined as {x,y}
^	FFT of a variable
$\text{Occlusion}_{\text{status}}$	Status of the tracker which informs if the tracker is occluded (True) or not occluded (False)

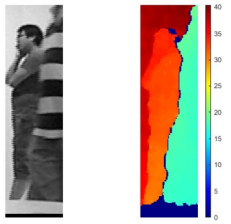
A more detailed version can be seen in Algorithm 1 of Fig. 7. Readers are to note that Fig. 7 has been split into four parts where Fig. 7a–c are components that help define the tracking algorithm in Fig. 7d, and hence can help understand the tracking algorithm better.

For the first frame, the tracker is provided with the ground truth. The tracker trains on this frame using the ground truth data (target position) to interpolate this position to the next frame. With this knowledge of the target location (from the previous frame), the tracker defines its search space around this target location, in the new frame. Search space is the area in which the tracker will attempt to locate the target as shown in Fig. 8.

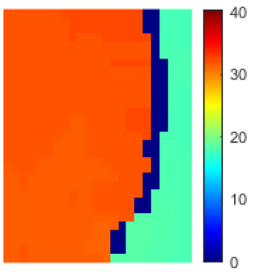
The tracker extracts all the possible patches from the space. The search space is dependent on the width of the tracker. If the tracker width is large, the search space is larger, and vice versa. If we keep the constant width of the search space, we might end up storing a much higher number of patches for



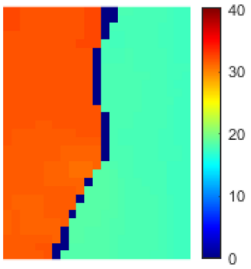
Frame 10: Grayscale and Depth



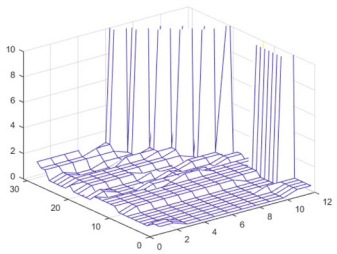
Frame 11: Grayscale and Depth



(a)
Centre depth patch:
Depth of the area around target patch center) extracted from the depth data of the target patch of frame 10



(b)
Centre depth patch:
Depth of the area around target patch center) extracted from the depth data of the target patch of frame 11



(c)
(After depth adjustment) Difference for center depth patches of two (occluding) Frames (Frame10 and Frame 11). Since the depth information changes significantly towards the center too, we see peaks in their difference.

Fig. 5 The figure shows how the difference in center depth patch changes for two subsequent frames when the target is occluded

smaller target sizes and very few target patches for large target sizes, making further computations difficult. The search space is also limited to the planar movements of the subject since we do not expect the target to move vertically in space as shown in Fig. 9.

For detection in the new frame, the tracker now correlates these patches (with gaussian correlation similar to KCF) with the initial target patch, to get all possible correlation responses. Mathematically, if z and x represent the features of the extracted patch and original target patch respectively, then we need to calculate $\widehat{k}^{xz}$ and $\widehat{\alpha}$ for fast training as represented by Eq. 2 (re-written below):

$$\widehat{f}(z) = k_f^{\widetilde{z}\widetilde{x}} \odot \widetilde{\alpha}_f,$$

where $\widehat{\alpha}$ is the model parameter used for model interpolation, obtained, and updated every time at the training stage as shown in Fig. 7. For gaussian correlation, we can write $\widehat{k}^{xz}$ as:

$$\widehat{k}^{xz} = FFT(\exp\Big(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|z\|^2 - 2F^{-1}\left(\widehat{x}^* \odot \widehat{z}\right)\right)\Big), \quad (3)$$

where $\odot$ is the dot product. $f(z)$ gives us the detection response and the maximum peak of this correlation response (highest correlation response) is observed as shown in Fig. 10.

If the maximum detection response is less than 50%, there are two possible scenarios: (a) tracker is partly visible (b) tracker is occluded as shown in Fig. 11. To confirm the status of the tracker, we compute the difference between the depth patches at frame $n-1$ and n (at the target centre), as shown in Fig. 5. If the difference in depth is minimal or close to zero, we know that target is not occluded, and the tracker will continue to track at the updated position computed using the detection score (maximum correlation response). However, if the difference in depth is high, we know the target is occluded. The tracker will, hence, stop tracking the target and will keep searching for the target in the search space. Once it locates the target (when a patch from the search space gives a

correlation response > 50%), it will re-detect the target and continue to track it.

4 Experimental setup

KCF tracker was originally implemented for RGB data. Hence, the dataset on which KCF was originally evaluated can not be used for testing the proposed RGB-D algorithm. Hence, to evaluate our algorithm, we use the Princeton RGB-D dataset. The dataset is also very diverse including examples of (a), occlusion (b), speed (c) and size (d) deformability which helps test the algorithm over a wide range of possible challenges. A sample image with RGB and its depth data can be seen in Fig. 12.

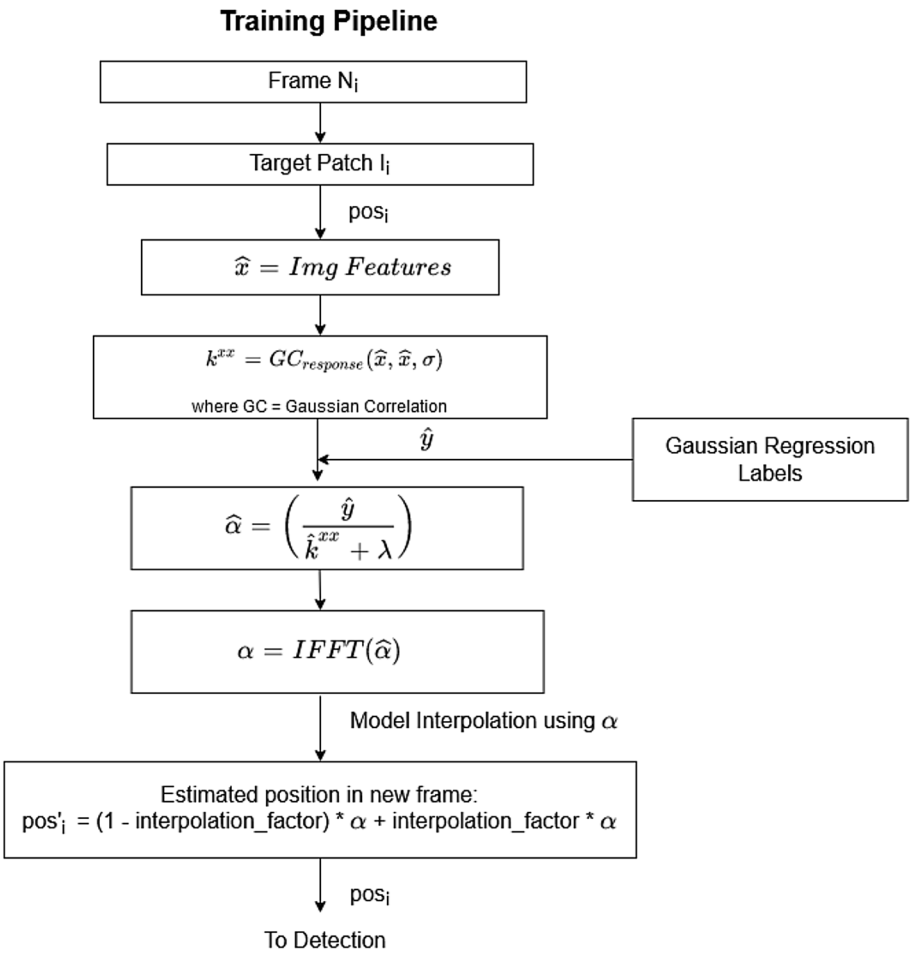
Apart from benchmark comparison (on the Princeton dataset), we collected our own RGB-D dataset for further evaluation. This dataset is collected using Microsoft Kinect V2. Figure 16 gives detailed information on the type of scenes and their respective samples.

To evaluate our method, we tested on two datasets (a) Princeton dataset and (b) the dataset collected by the authors using Kinect in real-time. We explain the setup of both the scenarios below.

4.1 Princeton dataset

In the Princeton RGB-D dataset, images are in 16-bit PNG format. Values at each pixel are the distance from Kinect to the object in mm. A sample image with RGB and its depth data can be seen in Fig. 12a and b. We can create a depth view of the sample image using the RGB-D data where x, y define pixel location and z the distance from the sensor. Depth is when an image is viewed as seen from z axis as shown in Fig. 12b. Depth view image helps us visualize the images as they are seen from the depth sensor.

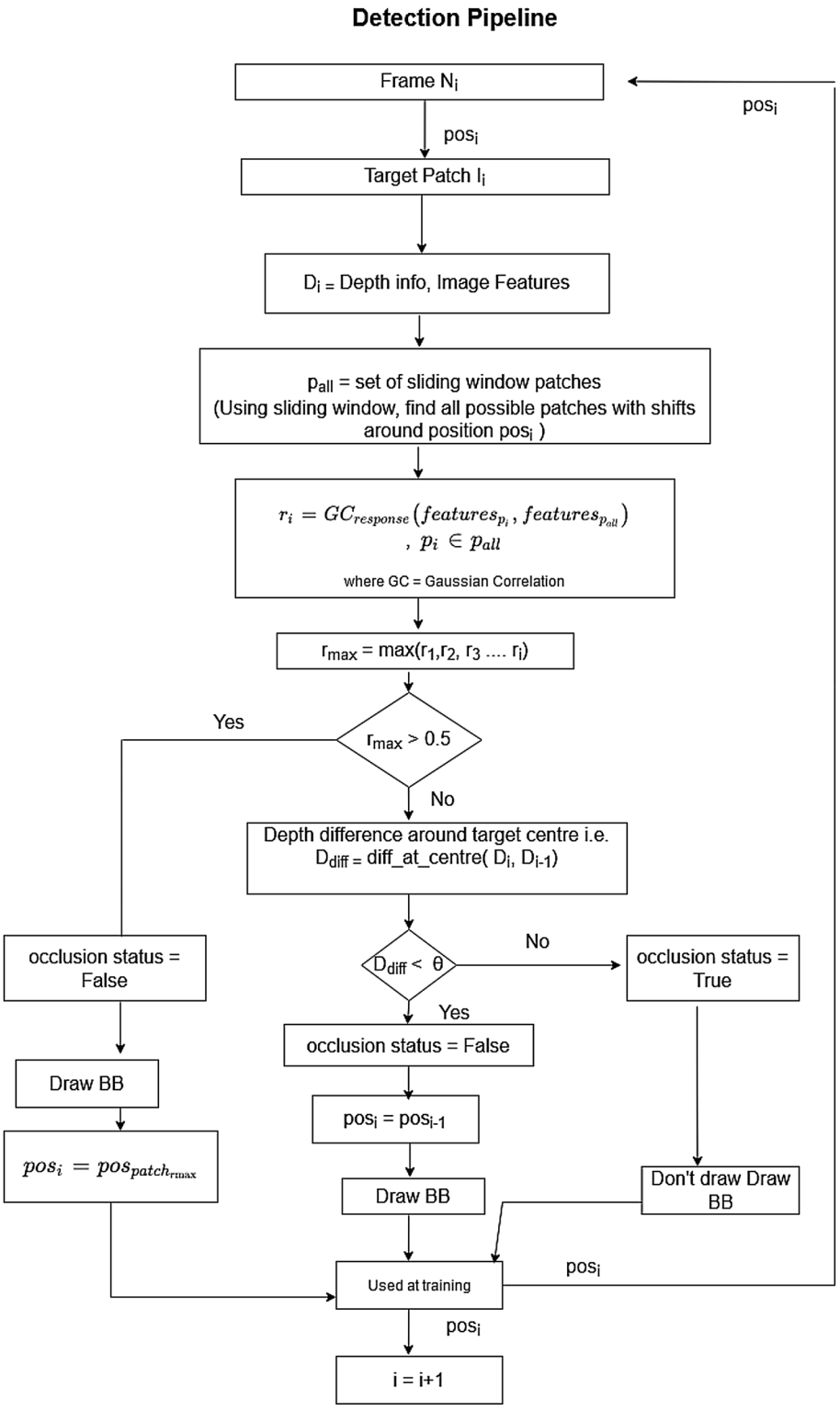
In this evaluation system, we use the Princeton data benchmark [23] to compare our tracker with KCF (our base tracker) and other trackers. This dataset uses 95 videos for evaluation. These datasets were originally captured using standard Microsoft Kinect 1.0. Due to Kinect's limit on the



(a) Tracking pipeline of the RGB-D tracker

Fig. 6 Flowchart of the training and detection pipeline of the RGB-D tracker

Fig. 6 (continued)



(b) Detection pipeline of the RGB-D tracker

kernel_correlation(x , z, σ)	
1	Calculate $\hat{k}^{xz} = FFT\ (\exp\ (\ -\frac{1}{\sigma^2}\ (\ \|\|x\|\|^2 +\ \|\|z\|\|^2 - 2F^{-1}\ (\hat{x}^* \odot\ \hat{z}\ ))$
2	Return: k = IFFT ($\hat{k}^{xz}$)

(a)

Training (x, y, λ)	
1	kernel_correlation((x , z, σ)
2	Compute $\hat{\alpha} = \left(\frac{\hat{y}}{\hat{k}^{xz} + \lambda}\right)$
3	Return $\alpha = IFFT(\hat{\alpha})$

(b)

Detection (x, z, λ)	
1	$\hat{k}^{xz}$ = kernel_correlation((x , z, σ)
2	Calculate $\hat{f}(z) =\ \hat{k}^{xz} \odot\ \hat{\alpha}$
3	Response = $f(z) = IFFT\ (\hat{f}(z))$
4	Return max ($f(z)$), the position of max ($f(z)$)

(c)

Algorithm 1: Tracker (x_{RGB}, x_{Depth})			
1	Extract patch $patch_{rgb}$ from image I_i		
2	Calculate D_i = Depth of image I_i		
3	*Training* ($features_x$, features_x, λ) where $features_x$ is the hog features of the target at pos_i. For frame 1, pos_i is provided by ground truth, for subsequent images is it is the output of the detection pipeline		
4	Interpolate the model with training output. Estimate target position at frame $i = i + 1$		
5	$i = i + 1$ at $t = t + 1$. At pos_i, find all possible patches p_{all} using a sliding window with shift s around the target		
6	Compute $gaussian\_correlation(\ \text{features}_{p_i}, \text{features}_{\text{p}_\text{all}}\ )$ for $p_i \in p_{all}$		
7	Find responses r_i of the classifier at all shifts s for p_{all}		
8	Find maximum response $r_{max}\ \ s.t.\ r_{max} \in\ r_s = \{r_1, r_2, r_3, \ldots r_{s_i}\}$		
9	Save (r_{max} , $patch_{r_{max}}$) where $patch_{r_{max}}$ is the patch which gives max response r_{max}		
10	**If**	$r_{max} > 0.5$	
11		Target detection using Detection($\text{features}_{\text{r}_{max}}$, $\text{features}_{\text{pos1}}$, λ) where $\text{features}_{\text{r}_{max}}$ = features of the patch which gives the maximum response, $\text{features}_{\text{pos1}}$ = features of the original patch	
12		Save position of max response	
13		$Occlusion_{status} = False$	
14	**else**	Calculate $D_{diff} = diff\_at\_centre\ (D_i, D_{i-1})$ where $diff\_at\_centre$ computes the difference of depths at the target's center (and adjusts the depth as required)	
15		**if**	D_{diff} < threshold
16			$Occlusion_{status} = False$; continue tracking (it may be a partly visible target), Go to Step 5
17		**Else**	$Occlusion_{status} = True$ (i.e. the target is occluded), no target detection, and no bounding box, Go to Step 5
18	If $i < N$; where N=total number of frames, go to Step 1 or else terminate.		

(d)

Fig. 7 Figure showing the RGB-D-based tracking algorithm. **d** Shows the full algorithm which uses modules shown in **a** the kernel correlation computation, **b** the training using the kernel correlation module, and **c** detection using the kernel correlation module

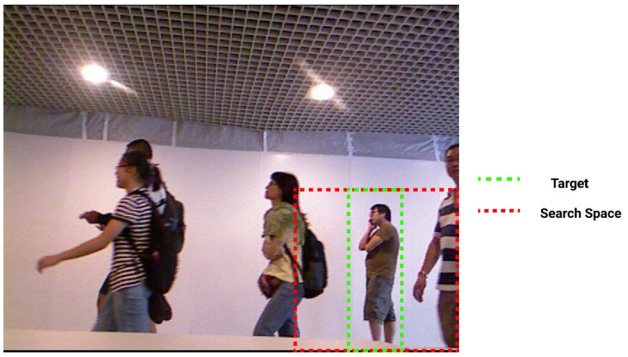


Fig. 8 The figure shows the target patch and associate search space of the tracking pipeline. Search space is dependent on **a** the target location in the previous frame and **b** the width of the target

minimum and maximum distance for accurate depth accuracy, these videos contain an indoor environment with object depth values ranging from 0.5 to 10 m.

This benchmark dataset has high diversity including single-tracking subjects like humans, animals, balls, etc. Figure 13 shows a few sample images from the Princeton dataset.

The various aspects of the dataset which show the diversity of samples are summarised as follows:

Target type: The dataset contains three types of objects: humans, animals, and rigid objects (example: toys and human faces which have the freedom to translate or rotate). Animal movements have out-of-plane rotation and some deformation. Tracking difficulty is expected o be slightly difficult for humans since the degree of freedom for human body motion is very high.

Scene type: Each scene has a different type of background. Some scenes like that of café and school are more complex with a lot of moving people moving around. Others like a turtle in a living room have a more static background.

Occlusion: Different possible occlusion scenarios are considered like, how long is the target occluded, whether target moves while being occluded or is static when it is occluded, etc.

Another criterion that was considered was the bounding box distribution over all sequences and over time. Hence, a target in a sequence does not necessarily be in the center but can be anywhere in the frame at any time. Readers are directed to [23] for a more detailed analysis of the sequence distribution of the dataset.

4.2 Real-time Kinect dataset

To further test our tracker, we tested it on the data we collected in real-time. Princeton data has a mix of subjects with a difference in scales, and color changes. We collected data on less complex scenarios (almost an ideal testing ground) with human data, minimal speed, and occlusion cases. Since we propose a long-term tracker that is expected to perform better in occlusion and out-of-scene scenarios, we focus particularly on such scenes. To accommodate for diversity in

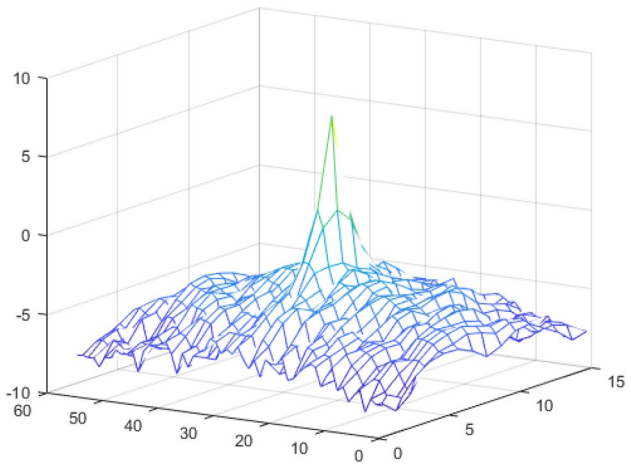


Fig. 10 Gaussian correlation of the base patch of the current frame with the patch from the previous frame

Fig. 9 The figure shows the issue if the search space is kept consistent across different images, it would lead to incorrect search space when the target size will vary

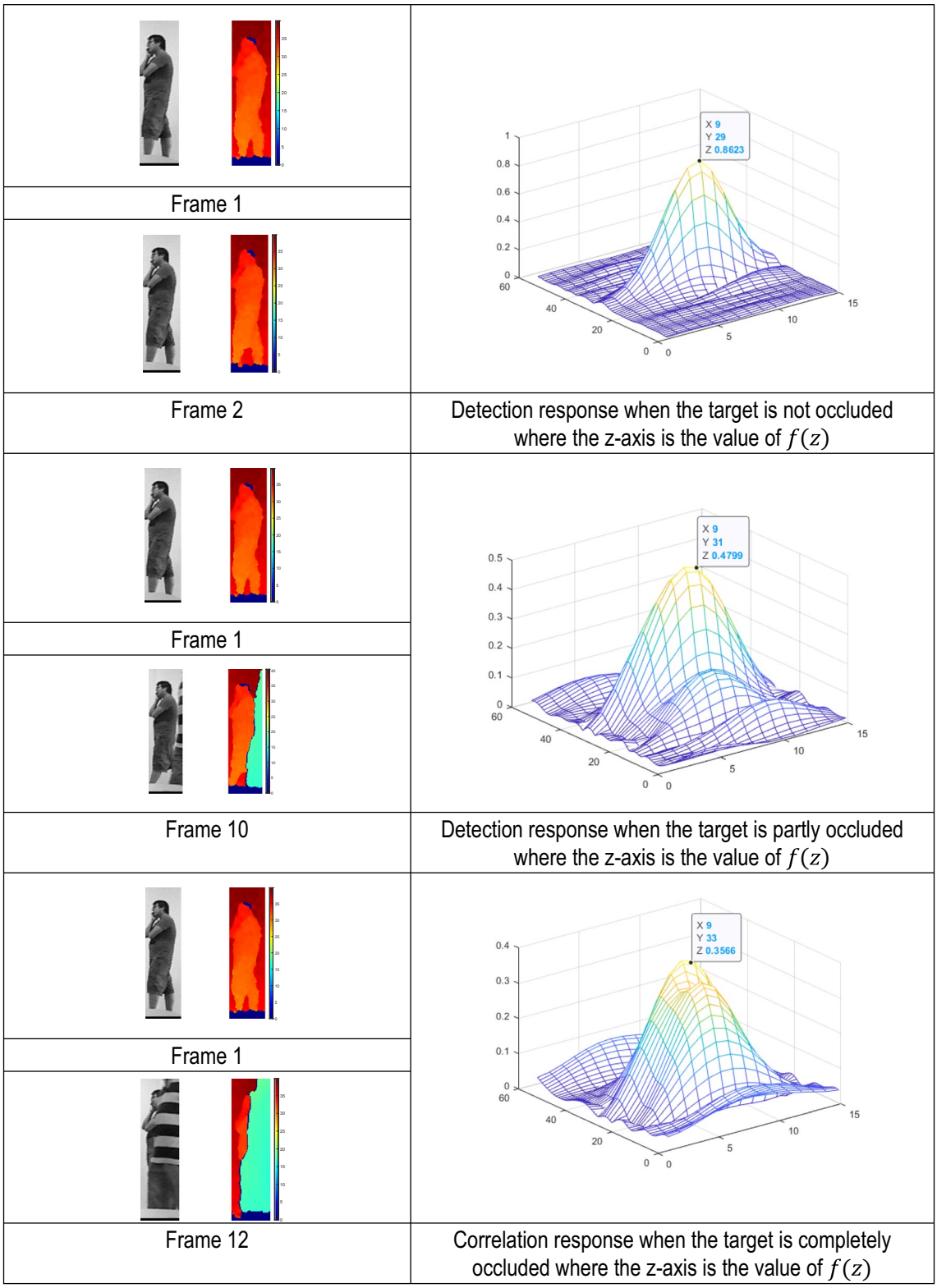


Fig. 11 Different detection responses of two subsequent frames with the non-occluding and occluding target. Note the peak value changes from 0.8 (i.e. 80%) when it is not occluded to 0.4 (40%) when it is partly occluded to 0.3 (i.e. 30%) when it is fully occluded

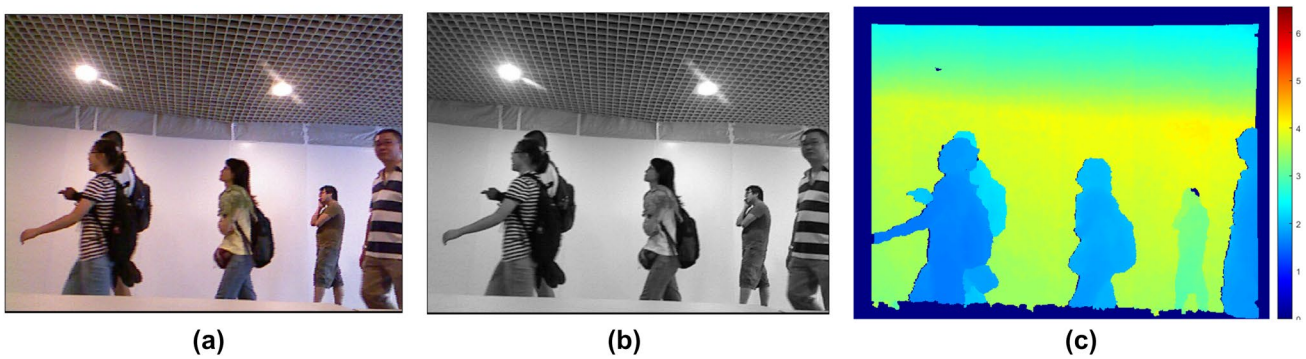


Fig. 12 **a** Sample RGB image from Princeton dataset. **b** Corresponding depth image, **c** depth colormap from the depth data

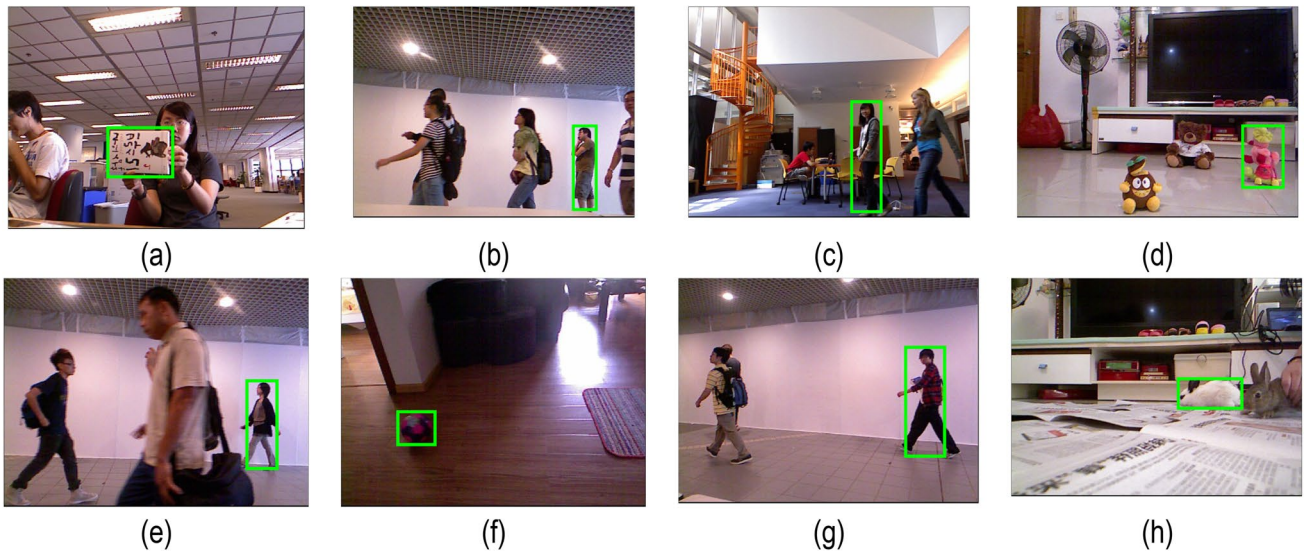


Fig. 13 The figure shows a few sample images with the target to be tracked (in green) from the Princeton Dataset. Best viewed in color

the dataset, we choose subjects that have different sizes and different types of occluding objects (chair, box, and human).

Data collection and hardware setup: The data is collected using Microsoft Kinect V2 in an indoor home environment. The subjects vary in size and gender to accommodate for diversity in subjects. The objects used for occlusion are things like a chair, a large box which is easily found in day-to-day life. The data collected assumes that the target is moving unidirectionally or bidirectionally in the horizontal plane. We do not consider scenarios where the target may tend to move towards and away from the sensor. They are useful in scenarios where the sensors are mounted in hallways (at a height at an angle) or mounted on stationary mobile robots which for the majority part observe the target moving from left to right or right to left. Hence, our data is a fair representation of various scenarios which are likely to occur in day-to-day lives. Due to COVID-19, there were few restrictions on who we can invite as subjects and where can we do the experiments. Despite these challenges, we made our best efforts to collect data in a reasonable setup. This data was collected using both Kinect for Linux and Kinect for MATLAB on Windows to accommodate for any software differences. There is a total of ~ 500 scenes with these different subjects and scenarios. Figure 14 shows a few sample RGB images and their corresponding depth images from our dataset.

Data annotation: Data were manually annotated by the author to collect the ground truth (GT). Each GT depicts tighest fitting bounding box that can be drawn to have the target within the box.

5 Result and observation

We evaluated the performance of our tracker on Princeton Dataset, a standard benbchmark for tracking and make observations of the results, in particular how the results are

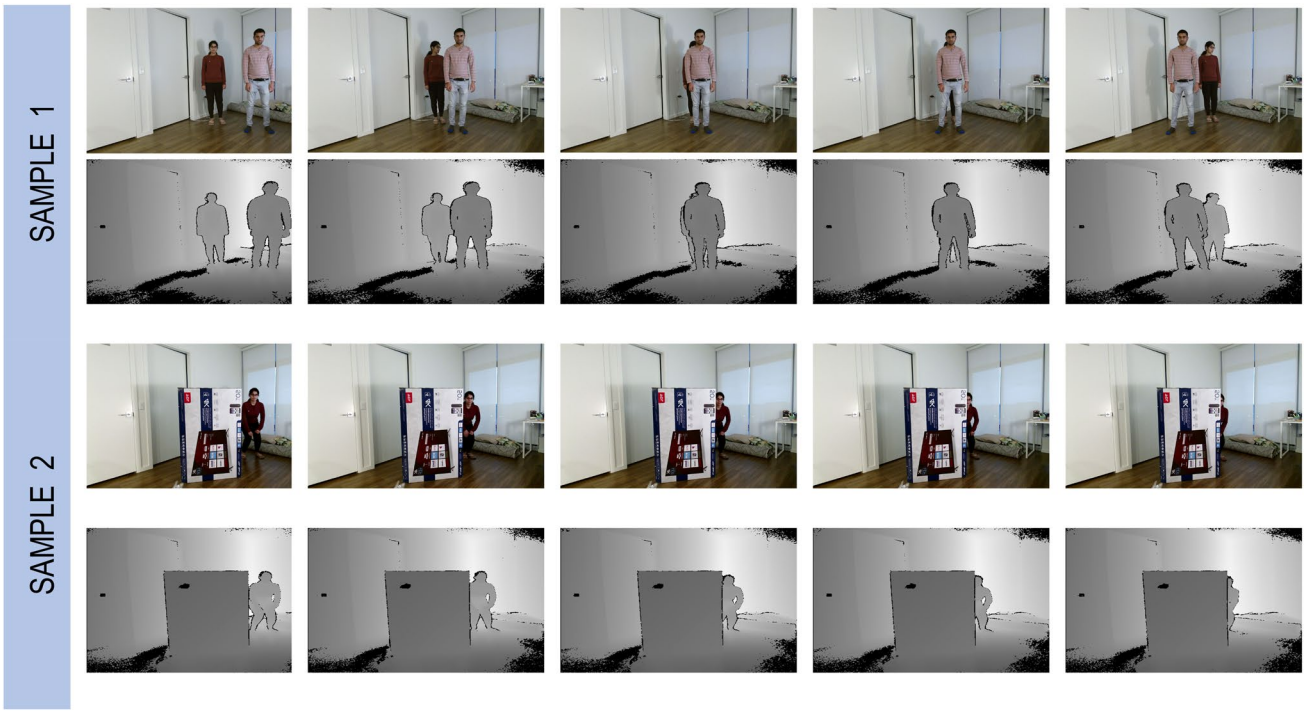


Fig. 14 Sample RGB and Depth images were collected for real-time analysis of the proposed long-term tracker

affected with human targets and theirt size. We also make detailed analysis when we test the algorithm on the dataset we collect using our own Kinect depth sensors and show that our proposed method ourperforms standard KCF tracker by a high margin.

5.1 Princeton dataset

5.1.1 Evaluation metrics

To evaluate the overall performance, we test it on Princeton data. It does not explicitly provide ground truth bounding box values but provides a script to evaluate results as detailed in [23]. We report the success rate provided by Princeton data evaluation (criterion used in the [24]) which is the ratio of overlap between the outputs and true bounding boxes:

$$r_i = \begin{cases} area\left(\mathrm{ROI}_{T_i} \cap \mathrm{ROI}_{G_i}\right) if both ROI_{T_i} and ROI_{G_i} exist \\ +1 if neither ROI_{T_i} and ROI_{G_i} exist \\ -1 otherwise \end{cases},$$

where ROI_{T_i} is the target bounding box in the i-th frame and ROI_{G_i} is the ground truth bounding box. By setting a minimum overlapping area r_t, they calculate the average success rate R of each tracker as follows:

$$R = \frac{1}{N}\sum_{i=1}^{N} u_i \text{where} u_i = \begin{cases} 1 if r_i > r_t \\ 0 otherwise \end{cases},$$

where u_i is an indicator denoting whether the output bounding box of the i-th frame is acceptable, N is the number of frames, and r_t is the minimum overlap area defining whether the output is correct. Since some trackers may produce outputs that have a small overlap ratio overall frames while others give large overlap on some frames and fail completely on the rest,r_t must be treated as a variable to conduct a fair comparison. Table 1 shows the success rate of our tracker as compared to KCF and other trackers under different categorizations on the Princeton dataset.

From the evaluation, we observe that our tracker performs best for human targets that are large in size and for scenarios where the target is occluded.

When it came to target type, it performs best for humans, with a success rate of 41%, close to Dhog (43% success rate) and KCF (39.7% success rate). The tracker performs worse for animals. One possible explanation is that tracker is unable to extract unique features from animals, especially since the color features of the animals in the dataset are very similar to the background (example: a white rabbit moving on a white floor as shown in Fig. 13). These animals are also very small making it harder for sufficient features to be extracted.

Table 1 The table shows the success rate (%) of our tracker on Princeton dataset

	Target type			Target size		Movement		Occlusion		Motion type	
	Human	Animal	Rigid	Large	Small	Slow	Fast	Yes	No	Passive	Active
Ours*	*41*	37	42	*46*	40	56	35	***47***	51	52	*46*
KCF [7]	39.7	***49.4***	*54.6*	40.1	***52.5***	*57.8*	42.9	35.2	***63.6***	***56.4***	43.7
Dhog* [23]	***43.3***	48.3	***55.9***	***47.2***	*50.3*	52.7	***47.5***	38.4	*63.5*	54.3	***46.9***
Struck [25]	35.4	47	53.4	45	43.9	***58***	39	30.4	*63.5*	54.4	40.6
VTD [26]	30.9	*48.8*	53.9	38.6	46.2	57.3	37.2	28.3	63.1	54.9	38.5
RGB [23]	26.7	40.9	54.7	31.9	46	50.5	35.7	34.8	46.8	*56.2*	33.7
CT [27]	31.1	46.7	36.9	39	34.4	48.6	31.5	23.3	54.3	42.1	34.2
PCflow* [23]	35.2	29.1	43.6	42.2	33.2	47.2	33.1	32.4	43.5	41.3	35.5
TLD [28]	0.29	0.35	0.44	0.32	0.38	0.52	0.30	0.34	0.39	0.50	0.31
MIL [29]	32.2	37.2	38.3	36.6	34.6	45.5	31.5	25.6	49	40.4	33.6
SemiB [30]	0.22	0.33	0.33	0.24	0.32	0.38	0.24	0.25	0.33	0.42	0.23
OF [23]	18	11	23	20	17	18	0.19	0.16	0.22	0.23	0.17

The evaluation compares the performance of our tracker against the KCF tracker and a few other trackers. Trackers marked with (*) use RGB and Depth data; others use only RGB data

One of the main objectives of our tracker is to be able to track when the target is occluded. Our tracker shows positive results in the 'Occlusion' category. We observe that, when the target is occluded, the success rate of our tracker is significantly better than KCF. KCF (RGB tracker) has a success rate of 35.2% whereas our tracker (RGB-D tracker) has 47%, a jump of approximately 12%, validating our hypothesis that depth data can significantly improve our results. Our trackers also perform well from a few other RGB-D-based trackers like Dhog which has a success rate of 38.4% and PCflow with a success rate of 32.4%.

In target size, our tracker performs best for larger targets (success rate of 46%) very close to Dhog, another RGB-D based tracker (success rate of 47.2%). Both of them are significantly better than KCF which uses only image features for tracking. When the object is large, not only features are easily and sufficiently captured, but the depth information is also large enough to track any significant changes in the depth of the target. Both these factors combined, make depth-based trackers better for larger objects.

It is observed that for 'Movement' and 'Motion Type', depth data can negatively affect the performance depending on the scenario. Depth information of an object will change significantly if an object rotates or move very fast. Image features are also adversely affected since (a) fast-moving objects will be blurry and (b) rotating objects will have different features at each frame (relative to the previous frame). Also, our tracker is not scale-invariant making the false positives even higher than other depth-based trackers in both these categories. Due to this reasoning, our current implementation of the tracker can not be generalized well on rotating objects and fast-moving objects (35% success rate as compared to 42% in KCF). Depth does help in identifying the target better, however, when the target is hidden or lost, the tracker attempts to locate the target in surrounding areas adding to a few false positives. For these reasons, during passive and active motions of the target, the performance of our tracker can be better or poor depending on a scenario as can be seen in Table 1. The current state of the art for occlusion detection on Princeton dataset is 3D-T [31] which provides a success rate of approximately 70% for occlusion scenarios and works best for humans (81%) and weakest for animals (64%). It uses 3D based detection that exploits sparse representation, object parts as well as adaptive particle sampling and pruning, all in a unified framework.

5.2 Real-time Kinect dataset

5.2.1 Evaluation metrics

With the implementation of a new RGB-D tracker, we aim to compare how well does new RGB-D tracker performs as compared to the KCF tracker (which only uses RGB). We take our inspiration for evaluation metrics from the original work of KCF which used average precision as their evaluation metrics. Additionally, we also use confusion matrices for two trackers (our RGB-D tracker and KCF RGB tracker) to discuss the tracker's ability to distinguish when the target is present (True Positives i.e., TP) and when it is absent (True Negatives i.e. TN).

Average precision: The first metric we use for comparison is average precision (AP). Precision can be stated as 'when the model guesses, how well does it guess correctly". Average Precision (AP) as the name suggests is calculated by taking the mean of the precisions (calculated for each dataset). If we define n as the number of scenes in the dataset (e.g., 5 scenes in our dataset) and s as the number of samples(images) in each scene in the dataset (e.g.,

Table 2 The table shows how elements of the confusion matrix (TP, FP, TN, FN) are computed for Method 1

True Positives (TP)	False Positives (FP)	True Negatives (TN)	False Negatives (FN)
The outcome when a model correctly predicts the positive class i.e. models says the target is present when the target is present	The outcome when a model incorrectly predicts the positive class i.e. models says the target is present when the target is absent	The outcome when a model correctly predicts the negative class i.e. models says the target is absent when the target is absent	The outcome when a model incorrectly predicts the negative class i.e. models says the target is absent when the target is present
$\text{TP(in\%)} = \frac{\text{TP}}{\text{Total}}$	$\text{FP(in\%)} = \frac{\text{FP}}{\text{Total}}$	$\text{TP(in\%)} = \frac{\text{TN}}{\text{Total}}$	$\text{TP(in\%)} = \frac{\text{FN}}{\text{Total}}$

Table 3 The table shows how elements of the confusion matrix (TP, FP, TN, FN) are computed for Method 2

For every time the target is predicted in the scene	How many times is the prediction correct? $\text{TP}_{\text{new}}\text{(in\%)} = \frac{\text{TP}}{\text{TP+FP}}$	How many times is the prediction incorrect? $\text{FP}_{\text{new}}\text{(in\%)} = \frac{\text{FP}}{\text{TP+FP}}$
For every time the target is not predicted in the scene	How many times is the prediction incorrect? $\text{FN}_{\text{new}}\text{(in\%)} = \frac{\text{FN}}{\text{FN+TN}}$	How many times is the prediction correct? $\text{TN}_{\text{new}}\text{(in\%)} = \frac{\text{TN}}{\text{FN+TN}}$

150 samples for scene 1, 180 samples for scene 2, etc.) such that total number of test samples is $N = n * s$, then,

$$\text{Precision}(P) = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

and

$$\text{AveragePrecision(AP)} = \frac{1}{N}\sum_{i=1}^{N} P,$$

where N is the number of samples in the dataset TP is True Positive and FP is False Positive.

Confusion matrices: Confusion matrices are a performance measure of a classification algorithm. It is a 2D array that compares the true labels against the predicted labels. It shows how well the predictions are made for any category. Confusion matrices can be defined in two ways as mentioned below.

Method 1: We can use a method where TP, FP, FN, TP are dependent on the actual 'number' of classifications w.r.t to the total number of classifications. Hence when Total = TP + FN + FP + TN, elements of the confusion matrix can be defined as shown in Table 2.

This representation though correct would not help much in our evaluation because for our base tracker KCF (RGB tracker) TN and FN will always be 0. It is because KCF can not identify the absence of the target. Intuitively, if the tracker does not have the ability to predict the absence of a target, our metrics should show that TN and FN can not be computed and hence '*can not be determined*'.

Method 2**:** Motivated by the belief that Method 1 may not be adequate, we modify our confusion matrix. We know that at any given instance of time, the target will be either predicted or not predicted. So, we can look at our predictions in every scene to see how well it matches our ground truth. We can now define our confusion matrix as (Table 3):

Table 4 The table shows the usefulness of Method 2 over Method 1 for two datasets

	Total no. of samples	No. of samples where the target was absent	No. of samples when the target was predicted absent	TN (%) (using Method 1)	TN (%) (using Method 2)
Dataset 1	100	20	20	20%	100%
Dataset 2	200	20	20	10%	100%

Table 5 Table shows the comparison of average precision (AP) in % when the two trackers are evaluated on the dataset

Tracker	AP (%)
Ours	66.35
KCF	39.99

This metric is also helpful when comparing and discussing datasets where the number of samples in each dataset varies. For example, consider the following scenario (Table 4):

Method 2 seems to be a fair comparison to answer *'how many false negatives were correctly predicted'* between two datasets with varying samples. Given the above motivation, we use Method 2 as the correct metric for evaluation and discussion.

Hence, for the KCF RGB tracker, TN and FN will always be '*undetermined*'. However, if our RGBD tracker performs well, including occlusion scenarios, we expect to have TN and TP values, both high in number.

Table 5 shows the comparison of average precision (AP) computed on our dataset. Figure 15 shows the confusion

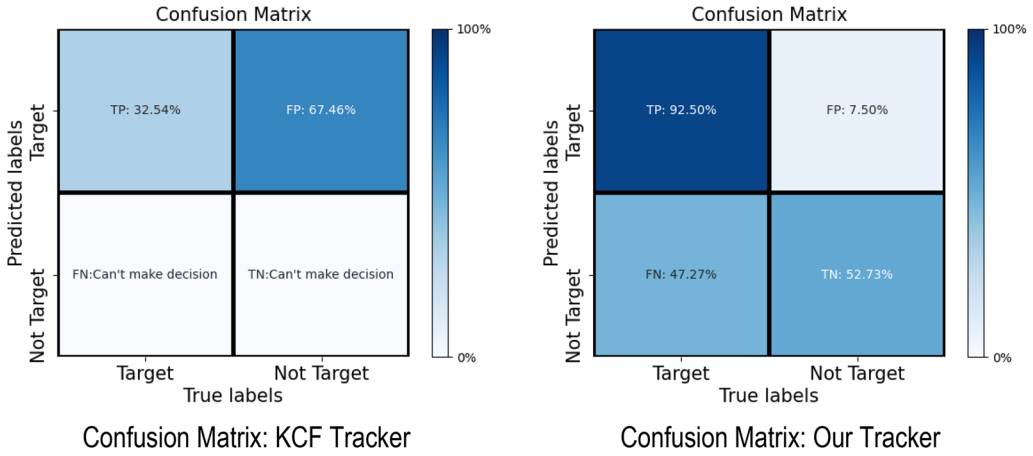


Fig. 15 Comparison of confusion matrix of test results between our RGB-D tracker and KCF RGB tracker. Note, each row of the confusion matrix adds up to 100% i.e. TP + FP = 100% and FN + TN = 100%. See the related section for an explanation

matrix for the test results for KCF RGB tracker and our RGB-D tracker. As discussed previously, we consider that TP + FP = 100% and FN + TN = 100%. We know that KCF RGB tracker gives no TN and FN, and hence, it can not make a decision on it. Figure 16 gives a detailed view of the samples in each scene along with the prediction made on them.

Hence by augmenting image features with the depth information, we make our tracker more robust to occlusions. It is able to detect occlusions and can also detect the absence of the target and stops tracking. This not only increases the *TN*, but also decreases *FP*. From Fig. 15. we can see that *TN* are 52.7% as compared to KCF where we cannot have this value. *TP* in our tracker is at 92.50% which is significantly higher than KCF at 32.54%. Figure 16 shows some sample results from our training method. However, we do have some failure cases which add to our *FP* and *FN* as we will now discuss.

5.2.1.1 Computation time KCF is the fastest with 141 frame-per-second (FPS) but it leads to less accurate results on average. The tracker achieves an average FPS of 10 on a PC with Intel i5 3.6 GHz CPU and 16 GB RAM on MATLAB with multi-threading. This is comparable to DS-KCF (30–43 FPS) but slightly lower due to the exhaustive search strategy we use to make accurate predictions. Our proposed method aims to improve the tracker in its success rate (Table 1) and precision of the tracker (Table 5) to make it less prone to errors during occlusion. Such trackers are suitable for real-time applications typically mobile robotic platforms. We estimate that with the use of GPUs, a higher FPS can be achieved due to more available compute resources and better processing.

5.2.1.2 Failure cases Adding depth information to existing features makes the tracker very robust to occlusions. However, when the target is absent, the tracker attempt to locate the target in the vicinity. Since the tracker keeps looking in the neighboring space, there are locations where it falsely predicts the target momentarily adding to False Positives. Also, the proposed tracker cannot detect partial appearance. Hence, in scenes where the target is coming out of occlusions, there are no clear detections adding to False Negatives. When the target is clearly out of the occlusion, the target is tracked again. Figure 17 shows some of the failure cases of the proposed tracker.

6 Discussions and Conclusions

In our proposed implementation, it is shown that depth cues when used with image features in KCF tracking can significantly improve the performance of the tracker. We observe that KCF tracker loses the target information once it is occluded adding to its inability to track a subject for a long time. Depth information, however, provided an additional layer of information that informs the tracker of the status of occlusion (true or false). Hence, it can stop tracking the target when the target is hidden. Using similar information, when the tracker is informed that the target is no more hidden, using the sliding window approach, it is able to search exhaustively in a large search space around the target to relocate the target. We are assuming the tracking happens in an indoor environment; hence, the nominal walking speed is considered. The experiments prove the usefulness of depth information as additional information in making the tracker robust. Experiments also show a few failure cases showing that at a certain time, the model may drift slightly from the expected path, possibly in cases (a) when the target is partly hidden (just coming of occlusion), (b) failed to correctly estimate the depth

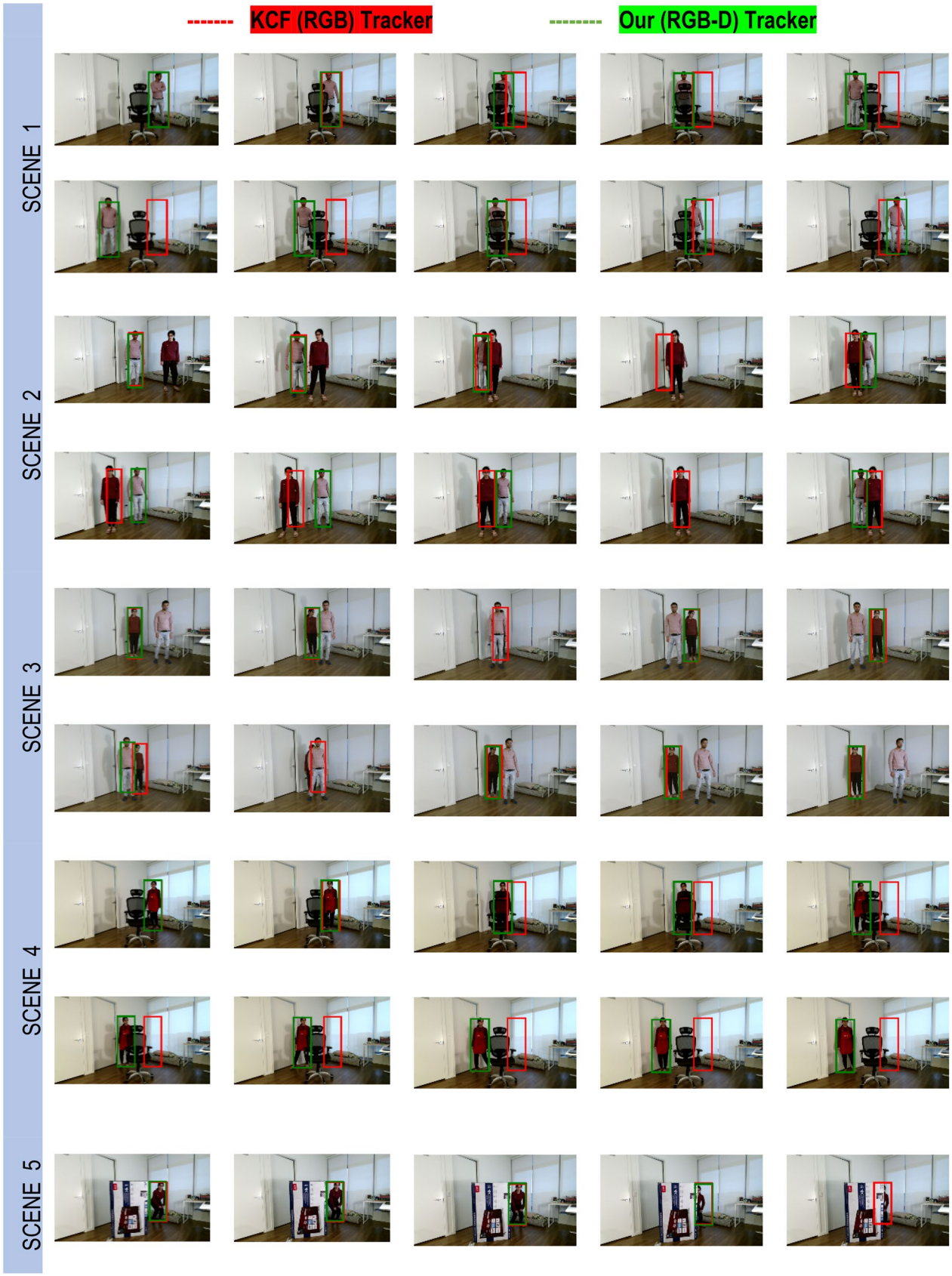
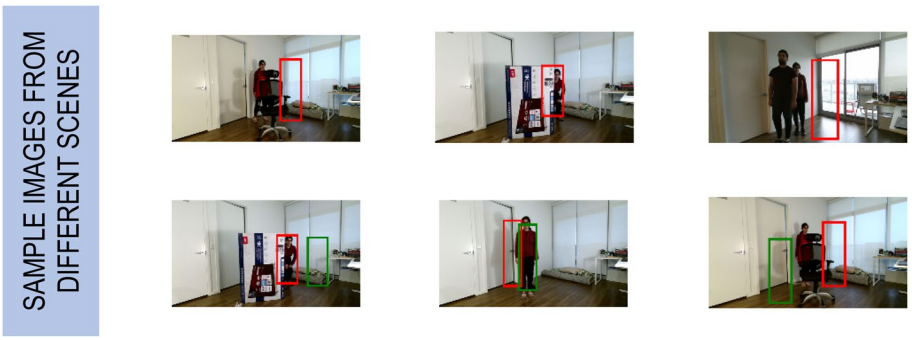


Fig. 16 Qualitative results for the proposed RGB-D tracker, compared to Kernelized Correlation Filter (KCF) tracker. Images show the tracking bounding box on test data. The red color denotes the KCF tracker and the green color denoted our tracker

Fig. 16 (continued)

Fig. 17 Failure cases of our RGB-D tracker



information. From our experimental observations, we see that the tracking performance of the tracker is improved compared to our KCF RGB tracker but also has the potential to improve even further.

Author contributions All authors have equal contributions.

Funding This work has been supported by the Natural Sciences and Engineering Research Council of Canada (Grant 31-611205).

Availability of data and material Please contact the author for data requests.

Declarations

Conflict of interest The authors declare that they have no competing interests.

Ethical approval and consent to participate Not applicable.

Consent for publication The picture materials quoted in this article have no copyright requirements, and the source, if applicable, has been indicated.

References

1. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.V.: XLNet: Generalized autoregressive pretraining for language understanding. arXiv. (NeurIPS):1–18 (2019)
2. Henriques, J.F., Caseiro, R., Martins, P., Batista, J.: High-speed tracking with Kernelized Correlation Filters. IEEE Trans Pattern Anal Mach Intell. **37**(3), 583–596 (2015)
3. Xie, Y., Lu, Y., Gu, S.: RGB-D object tracking with occlusion detection. In Proceeding of - 2019 15th International Conference of Computer Intelligent Secure CIS. 2019;11–15 (2019)
4. Qian, Y., Lukežič, A., Kristan, M., Kämäräinen, J.-K., Matas, J.: DAL—a deep depth-aware long-term tracker. 2019; Available from: http://arxiv.org/abs/1912.00660
5. Hannuna, S., Camplani, M., Hall, J., Mirmehdi, M., Damen, D., Burghardt, T., et al.: DS-KCF: a real-time tracker for RGB-D data. J. Real-Time Image Process **16**(5), 1439–1458 (2019)

6. Bolme, D.S., Beveridge, J.R., Draper, B.A., Lui, Y.M.: Visual object tracking using adaptive correlation filters. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2544–2550 (2010)
7. Henriques, J.F., Rui, C., Pedro, M., Horge, B.: Kernelized Correlation Filters **37**(3), 583–596 (2015)
8. Yadav, S., Payandeh, S.: Understanding tracking methodology of Kernelized Correlation Filter. In: 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2018. IEEE. pp. 1330–1336 (2019)
9. Yadav, S., Payandeh, S.: Real-time experimental study of Kernelized correlation filter tracker using RGB Kinect camera. In: 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2018. IEEE. pp. 1324–1329 (2019)
10. Yadav, S., Payandeh, S.: Critical overview of visual tracking with kernel correlation filter. Technologies **9**(4), 93 (2021)
11. Montero, A.S., Lang, J., Laganière, R.: Scalable kernel correlation filter with sparse feature integration. In Proceeding of the IEEE International Conference on Computer vision. 2016-Febru:587–594 (2016)
12. Danelljan, M., Häger, G., Khan, F.S., Felsberg, M.: Accurate scale estimation for robust visual tracking. In: BMVC 2014—Proceedings of the British Machine Vision Conference 2014. (2014)
13. Danelljan, M., Hager, G., Khan, F.S., Felsberg, M.: Learning spatially regularized correlation filters for visual tracking. In Proceedings of the IEEE Conference on Computer vision. 2015 Inter:4310–4318 (2015)
14. Danelljan, M., Robinson, A., Khan, F.S., Felsberg, M.: Beyond correlation filters: learning continuous convolution operators for visual tracking. Eccv **5**, 1–9 (2016)
15. Zuo, W., Member, S., Wu, X., Lin, L., Member, S.: Learning support correlation filters for visual tracking. IEEE Trans. Pattern Anal. Mach. Intell. **41**(5), 1158–1172 (2019)
16. Zhang, K., Zhang, L., Liu, Q., Zhang, D., Yang, M.H.: Fast visual tracking via dense spatio-temporal context learning. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics). 8693 LNCS(PART 5):127–141 (2014)
17. Liu, T., Wang, G., Yang, Q.: Real-time part-based visual tracking via adaptive correlation filters. In: Proceeding of IEEE Conference on Computer Vision and Pattern Recognition. 07–12-June:4902–4912 (2015)
18. Zhang, T., Xu, C., Yang, M.H.: Multi-task correlation particle filter for robust object tracking. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 (2017)
19. Camplani, M., Hannuna, S., Mirmehdi, M., Damen, D., Paiement, A., Tao, L, et al.: Real-time RGB-D tracking with depth scaling Kernelised correlation filters and occlusion handling. 145.1–145.11 (2015)
20. Hou, N.A., Zhao, X.-G., Zeng-Guang.: Online RGB-D tracking via detection-learning-segmentation. Pattern Recognit (ICPR), 2016 23rd Int Conf. 1231–1236 (2016)
21. Rasoulidanesh, M., Yadav, S., Herath, S., Vaghei, Y., Payandeh, S.: Deep attention models for human tracking using RGBD. Sensors (Switzerland) **19**(4), 750 (2019)
22. Babenko, B., Yang, M.H., Belongie, S.: Robust object tracking with online multiple instance learning. IEEE Trans Pattern Anal Mach Intell. **33**(8), 1619–1632 (2011)
23. Song S, Xiao J. Tracking revisited using RGBD camera: Unified benchmark and baselines. In: Proceeding of IEEE International Conference on Computer Vision 233–240 (2013)
24. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes (VOC) challenge. Int J Comput Vis. **88**(2), 303–338 (2010)
25. Hare, S., Golodetz, S., Saffari, A., Vineet, V., Cheng, M.M., Hicks, S.L., et al.: Struck: structured output tracking with kernels. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 2096–2109 (2016)
26. Kwon, J., Lee, K.M.: Visual tracking decomposition. In Proceeding of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1269–1276 (2010)
27. Zhang, K., Zhang, L., Yang, M.H.: Fast compressive tracking. IEEE Trans. Pattern Anal. Mach. Intell. **36**(10), 2002–2015 (2014)
28. Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-learning-detection. IEEE Trans. Pattern Anal. Mach. Intell. **34**(7), 1409–1422 (2011)
29. Babenko, B., Belongie, S., Yang, M.H.: Visual tracking with online multiple instance learning. In: 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Work CVPR Work 2009. 2009 IEEE:983–990 (2009)
30. Grabner, H., Leistner, C., Bischof, H.: Semi-supervised on-line boosting for robust tracking. (813399):234–247 (2008)
31. Bibi, A., Zhang, T., Ghanem, B.: 3D part-based sparse tracker with automatic synchronization and registration. In Proceeding of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2016-Decem:1439–1448 (2016)