J Real-Time Image Proc
DOI 10.1007/s11554-016-0654-3

ORIGINAL RESEARCH PAPER

DS-KCF: a real-time tracker for RGB-D data

Sion Hannuna[1] · Massimo Camplani[1] · Jake Hall[1] · Majid Mirmehdi[1] · Dima Damen[1] · Tilo Burghardt[1] · Adeline Paiement[1] · Lili Tao[1]

Received: 11 May 2016 / Accepted: 11 November 2016

Abstract We propose an RGB-D single-object tracker, built upon the extremely fast RGB-only KCF tracker that is able to exploit depth information to handle scale changes, occlusions, and shape changes. Despite the computational demands of the extra functionalities, we still achieve real-time performance rates of 35–43 fps in MATLAB and 187 fps in our C++ implementation. Our proposed method includes fast depth-based target object segmentation that enables, (1) efficient scale change handling within the KCF core functionality in the Fourier domain, (2) the detection of occlusions by temporal analysis of the target's depth distribution, and (3) the estimation of a target's change of shape through the temporal evolution of its segmented silhouette allows. Finally, we provide an in-depth analysis of the factors affecting the throughput and precision of our proposed tracker and perform extensive comparative analysis. Both the MATLAB and C++ versions of our software are available in the public domain.

Keywords RGB-D tracking · Correlation filters · Scale and shape changes handling · Occlusion detection · Depth-based segmentation

✉ Majid Mirmehdi
M.Mirmehdi@bristol.ac.uk

Sion Hannuna
sh1670@bristol.ac.uk

Massimo Camplani
massimo.camplani@bristol.ac.uk

Jake Hall
J.Hall@bristol.ac.uk

Dima Damen
D.Damen@bristol.ac.uk

Tilo Burghardt
tb2935@bristol.ac.uk

Adeline Paiement
A.Paiement@bristol.ac.uk

Lili Tao
L.Tao@bristol.ac.uk

[1] Visual Information Laboratory, Faculty of Engineering, University of Bristol, Bristol, UK

1 Introduction

One of the most fundamental and active areas of computer vision is object tracking, as demonstrated by several recent reviews [30, 38, 43] and challenges [27]. These show that tracking algorithms based on 2D visual cues (hereafter referred to as RGB trackers) have greatly improved, but that the tracking problem is still far from being resolved. A number of ongoing challenges still remain, such as handling object appearance changes, illumination variations, occlusions, shape deformations, and camera motion.

On the other hand, depth sensors have seen a recent surge in popularity and have been successfully used in many computer vision applications [20]. These devices facilitate the acquisition of reliable depth data in indoor environments at no extra computational cost. Depth information can potentially be exploited to boost the performance of traditional object tracking algorithms. Research into combining depth and colour data for tracking is still in its infancy [7]. However, it has been demonstrated that state-of-the-art RGB tracking algorithms can be outperformed by approaches that fuse colour and depth, for example [6, 7, 15, 36, 39, 41].

To expand on this concept, this paper will first review in detail a number of recent object tracking algorithms based on the combination of colour and depth data. Note that we

Published online: 24 November 2016

shall refer to these methods as RGB-D trackers. Then, the RGB-D method proposed in this paper will be presented which is based on, and improves upon, the RGB kernelized correlation filter (KCF) tracker from [22]. The KCF tracker combines high accuracy and fast processing speeds as demonstrated in [22] and elsewhere, and in particular in the benchmarking work in [43], where processing rates exceeding 150 fps were reported. Despite the KCF tracker not being able to handle changes in scale or shape, it still obtains performances comparable with other state-of-the-art approaches (it is in the eighth position in the top ten rank in [43]), while guaranteeing a very high processing throughput. Unsurprisingly, given these results, KCF has quickly gained popularity in the computer vision community as demonstrated by several recently published works and extensions [6, 9, 11, 33, 35].

In this work, we provide a detailed description and evaluation of our real-time RGB-D tracker, namely DS-KCF, which improves on the performance of the KCF tracker by (1) integrating depth and colour features in the KCF framework, and efficiently handling (2) scale changes, (3) occlusions, and (4) aspect ratio changes of the target model. We reported an early version of DS-KCF in [6] which operated the first two of the above functionalities, but here we provide a more detailed presentation and evaluation of our tracker, including its full functionalities. We exploit the target's depth distribution to identify and efficiently model its scale changes in the Fourier domain. We handle occlusions by identifying sudden changes in the target region's depth histogram and recovering lost tracks by searching for the unoccluded object in specifically identified key areas. Finally, temporal evolution of the segmented target's silhouette is evaluated to identify shape changes. Another new aspect of the DS-KCF tracker is that we have added a Kalman filter motion model for improved tracking performance during occlusions. Our DS-KCF tracker can run at an average rate of 35.7 fps (increasing to 43.5 fps without shape change handling if it is not necessary) implemented in MATLAB and at an average rate of 187 fps (without shape change handling) implemented in C++, measured on 100 sequences of the Princeton RGB-D dataset [39]. To the best of the authors' knowledge, DS-KCF is the fastest single-object RGB-D tracker released so far.

The paper is structured as follows. In Sect. 2, state-of-the-art tracking methods based on the fusion of colour and depth information are reviewed. In Sect. 3, the basic features of the KCF tracker are described. In Sect. 4, the proposed DS-KCF tracker and its novel contributions are reported. Experimental results are presented in Sect. 5, and our conclusions are in the last section.

2 Past contributions in RGB-D tracking

Visual tracking combined with accurate, dense depth data has only recently gained the attention of the computer vision community due to the availability of low-cost RGB-D sensors. Hence, relatively few significant works have been published to date.

One of the first contributions in this area, where the importance of depth data in tracking algorithms was clearly demonstrated, was in [39]. Its authors released a large public dataset which includes more than 100 videos (Princeton dataset, more details in Sect. 5). HoG features calculated on both colour and depth data were used to build the target model, with negative instances obtained by randomly sampling the rest of the image. These features were then used by an SVM classifier to detect the object along the sequence. SVM detections were linearly combined with the outcome of an optical flow tracker to improve the tracking reliability. Occlusion was also handled to a considerable extent in [39], based on the assumption that the depth distribution of the plane closest to the camera contains the tracked object. During occlusion, optical flow tracks the object occluding the target until the target re-emerges. The overall approach was computationally expensive at 0.26 fps on average (for the dataset in [39]), due to the exhaustive search, optical flow computations, and elaborate colour and depth segmentation. However, in terms of precision it outperformed state-of-the-art RGB-only trackers.

A similar approach was presented in [10], where the RGB tracker TLD [24] was extended to include depth as an additional feature in the tracking stage. The depth information was also employed to filter out background pixels and to detect occlusions with the same approach proposed in [39]. Also, the ratio between depth and object size was used to refine the TLD detection phase. This method reported a processing rate of less than 10 fps.

In [41] a multi-cue framework based on a combination of optical flow, colour, and depth data was presented. Optical flow (estimated on RGB images) was used to roughly estimate the target's movements, and the target position was then refined by partitioning its area into four subregions: top, down, left, and right. For each subregion, a colour histogram model and a depth model, represented by the mean depth, were generated. The distances between the colour models and the depth models were then estimated and linearly combined, and the new target position was assigned to the region that minimizes that distance. Their work did not detail how their model was updated, and their results were reported on sequences not publicly available without data about the processing throughput.

The algorithm presented in [15] extended the condensation-based RGB tracker of Klein and Cremers [26] to incorporate depth data and predict the 3D spatial state of the particles in the condensation algorithm. Their boosting classifier was built from a pool of greyscale, colour, and depth features which was kept small to provide the right balance between computational efficiency and accuracy. Tracking was adaptive as their classifier was retrained with tracked examples. Occlusions were detected when the tracker response was below a certain threshold. They reported an average processing rate of 30 fps.

The RGB-D tracker presented in [36] based on a colour- and depth-based 'occlusion aware' particle filter tracking framework was introduced. A particle represents a region's bounding box and an occlusion threshold. When this threshold is breached, occlusion is detected and the bounding box search area is expanded. The proposed particle filter system utilizes a combination of different features, including histograms of colour and depth, a histogram of texture, 3D shape parameters, and edge information. This method is the best RGB-D tracker tested on the Princeton dataset [39] in terms of accuracy; however, it performs at an average processing rate of 1 fps.

In [2], a *depth-only* tracker was proposed by adapting the Struck RGB tracker [21] to depth data by introducing a new feature called local depth pattern for tracking (LDPT), inspired by the local binary pattern feature. Their approach outperforms RGB trackers, but its performance is poor compared to other RGB-D trackers on the Princeton dataset [39]. No data about the processing rate were reported.

The DS-KCF tracker we introduced in [6], and outline in more detail and analysis in Sect. 4, provides a more balanced solution than other RGB-D trackers, as it combines strong localization accuracy with real-time performance. Next, we outline the fundamentals of the KCF tracker, followed by a detailed presentation of DS-KCF and its features.

3 The KCF tracker and its extensions

In this section, we firstly describe the KCF tracker with focus on aspects pertinent to our proposed extensions. For a more detailed description, with associated proofs, refer [22]. Additionally, refer the works presented in [3, 9, 14, 22, 23] for a comprehensive overview of correlation filters. Then, we provide a short review of the most relevant extensions to the KCF tracker, with more details found in the recent survey [9]. Table 1 contains a summary of the key symbols used throughout the paper.

KCF tracker Henriques et al. [22] proposed using the 'kernel trick' [37] to extend correlation filters for very fast RGB tracking. Their so-called KCF tracker is noteworthy for combining high accuracy and processing speed. In the comprehensive RGB tracking benchmarking work in [43], KCF was ranked in the eighth position while coming in first as the fastest. Furthermore, its accuracy is comparable to other state-of-the-art approaches. These attributes make KCF a method of choice for those who need a very fast and reliable tracker.

Table 1 Summary of used symbols

Symbol	Description
$\mathbf{z}$	Tracked patch area
Ω	Segmented region
Γ	Spatial extent of Ω
μ, σ	Mean depth and standard deviation of Ω
$\mathcal{S}^r, \mathcal{S}^q$	Continuous and discrete scaling factors

As shown in Fig. 1, the processing pipeline in the KCF tracker is comprised of training, detection, and retraining at the new target location. This final step includes updating the previous frame's model parameters with the current one based on an empirically determined interpolation factor.

KCF exploits the properties of circulant matrices to achieve efficient learning. An $m \times m$ circulant matrix $C(\mathbf{x})$ may be constructed from an $m \times 1$ vector, x, by applying a cyclic shift operator

$$C(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_m \\ x_m & x_1 & x_2 & \dots & x_{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \dots & x_1 \end{bmatrix}. \quad (1)$$

Circulant matrices are pertinent to tracking and detection as they implicitly encode convolution. Given that $C(\mathbf{x})\mathbf{y}$ is the convolution of $\mathbf{x}$ and $\mathbf{y}$, convolution may be performed in the Fourier domain via element-wise multiplication, i.e.

$$C(\mathbf{x})\mathbf{y} = \mathcal{F}^{-1}(\mathcal{F}^*(\mathbf{x}) \odot \mathcal{F}(\mathbf{y})), \quad (2)$$

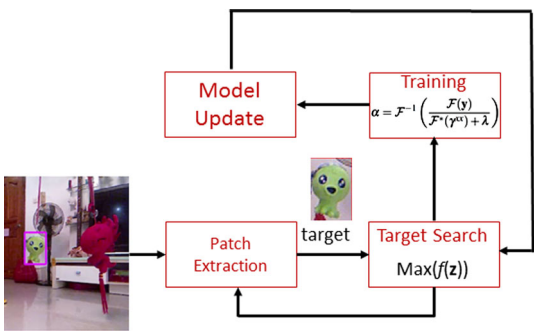

Fig. 1 Block diagram of the KCF tracker

where $*$ is the complex conjugate, $\odot$ is the element-wise multiplication, and $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier and inverse Fourier transformations.

Applying the above ideas to linear regression, it may be shown that the standard formulation for ridge regression, where X is the design matrix (X^{H} is the Hermitian transpose), λ is a penalty term, and $\mathbf{y}$ is the regression target, is

$$\mathbf{w} = (X^{\mathrm{H}}X + \lambda I)^{-1}X^{T}\mathbf{y}, \tag{3}$$

and this may be reformulated as

$$\mathcal{F}(\mathbf{w}) = \frac{\mathcal{F}^{*}(\mathbf{x}) \odot \mathcal{F}(\mathbf{y})}{\mathcal{F}^{*}(\mathbf{x}) \odot \mathcal{F}(\mathbf{x}) + \lambda}, \tag{4}$$

where the fraction indicates element-wise division.

The efficiency of the KCF is apparent from (4), as it reduces the computational complexity of the general regression problem from $O(n^3)$ for ridge regression to the element-wise operations included in (4). To completely recover the values of $\mathbf{w}$ we also have to consider the DFT complexity equal to $O(n \log n)$.

Similar expressions can be derived when nonlinear regression is used. By using the kernel trick, the solution $\mathbf{w}$ may be stated as $\mathbf{w} = \sum_i \alpha_i \varphi(\mathbf{x}_i)$ where the variables α_i need to be estimated instead of $\mathbf{w}$, and φ is the function that maps $\mathbf{x}$ into the nonlinear feature space. The kernelized version of ridge regression can be written as

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1}\mathbf{y}, \tag{5}$$

where K is the circulant kernel matrix, with elements K_{ij} corresponding to $\gamma(x_i, x_j)$ and γ is the selected kernel function. As in [22], (5) can be reformulated using similar circulant matrices concepts such that

$$\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{y})}{\mathcal{F}^{*}(\gamma^{xx}) + \lambda}\right), \tag{6}$$

where γ^{xx} is the first row of K. We refer to (6) generally as the training phase of the KCF tracker, as reported in the second module in Fig. 1. Once $\boldsymbol{\alpha}$ is calculated, the circulant matrix properties can be exploited to estimate efficiently and simultaneously the response of the classifier $f(\mathbf{z})$ at various image patches $\mathbf{z}$,

$$f(\mathbf{z}) = \mathcal{F}^{-1}(\mathcal{F}^{*}(\gamma^{xz}) \odot \mathcal{F}(\boldsymbol{\alpha})). \tag{7}$$

This operation is performed in the Fourier domain, employing element-wise multiplication and the DFT. The new target location is then selected as the one that maximizes the response $f(\mathbf{z})$ (see first block of Fig. 1).

In [22], different kernels were presented, with the Gaussian kernel demonstrating optimal trade-off between accuracy and computational complexity (equal to $O(n \log n)$). Hence, in our implementation we use

$$\gamma^{xz} = \exp\left(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|z\|^2 - 2\mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}^{*}(z))\right)\right). \tag{8}$$

To summarize, the KCF tracker is based on a simple processing chain. An image patch is extracted at the estimated target location, and a precomputed cosine window is applied to the patch to reduce the noise in the Fourier domain. The target position is detected by maximizing $f(\mathbf{z})$ [as in (7)]. The model is trained using Gaussian-shaped regression targets that provide smooth responses. Model updates are performed by linearly interpolating the new parameters $\boldsymbol{\alpha}$ and $\mathbf{x}$ with the current ones.

KCF assumes the bounding box containing the target object is initialized once at the beginning of the sequence. This assumption has been accepted to evaluate all single-object trackers, as for example in recent reviews in [30, 38, 43] and challenges [27]. KCF-based trackers can be initialized by a number of detectors, and to the best of the authors' knowledge the study of an optimal combination between a detector and KCF-based tracker has not yet been explored.

KCF extensions Due to the unique characteristics of the KCF, several extensions have been proposed since [22]. In this section, we report the ones which share characteristics with our proposed modifications (i.e. change of scale, feature selection, etc.). For a more detailed review of KCF variations, see [9].

Different feature sets have been used within the KCF framework, and as expected, the dimensionality of the selected features has a direct impact on the time complexity and memory requirements of their respective trackers. Both raw RGB and HoG features were successfully tested in the original KCF paper [22]. HoG features provide a higher accuracy without compromising real-time performance as the higher dimensionality of the feature space is compensated for by smaller patch sizes due to HoG's discrete cells. In [12], an exhaustive evaluation of the KCF tracker with different colour features is provided. The authors demonstrate that colour name (CN) combined with the intensity channel guarantees the best tracking accuracy. Additionally, they provide an adaptive feature reduction approach to maintain a high processing speed (around 100 fps). CN has also been used in [31, 44] combined with HoG features. To the best of the authors' knowledge, the proposed DS-KCF and eDS-KCF frameworks are the only ones fusing both colour and depth data in a *KCF-like* framework.

One of the main drawbacks of the KCF tracker is that it is not able to cope with scale changes. Specifically, the target and model size are fixed according to the initial object size. Many of the extensions proposed so far try to solve this problem by using a pool of scale models. In [11]

the KCF position and size were assigned to the solution that maximizes Eq. (7) among the different scales and positions. The authors demonstrated that while this exhaustive search provides the most accurate results, it leads to huge computational demands. They proposed an optimized solution by separately finding the new position as in the KCF and then estimating the scale at the position. In this way, they obtained a good balance between speed and accuracy, reaching 24 fps. A very similar strategy was used in [13, 48]. The same base idea has also been proposed by [31], where target position and the current target scale were selected as the one corresponding to the maximum response value among all the scales. In this case, to reduce the computational complexity, all the model templates and processed patches are rescaled to the original target size such that only one model is constantly estimated and updated. However, the system can reach a processing rate of only 7 fps. The same approach is applied in [35].

Patch-based approaches have also been used to cope with scale changes. In [45], four patch trackers based on the KCF modification proposed in [12] were employed. Variations of the patches' sizes were estimated, as in [11], and combined in one single measure to evaluate the entire object's scale variation. A similar idea was proposed in [32] where different KCF trackers were combined in a Monte Carlo framework. While these methods could be effective and improve the KCF performance, their main drawback is that the computational complexity grows linearly with the number of patches used.

Finally, in [47], a scale change was detected by analysing the motion flow among consecutive frames, forcing the tracker to be reinitialized. As we will show in the following section, the use of depth data is fundamental to obtain an accurate and efficient estimation of scale changes.

As far as managing occlusions is concerned, only the work presented in [33] has considered it, where a part-based tracker was employed. When the level of response of each part tracker was below a certain threshold, the model of the corresponding part was not updated and that object's part was considered occluded. The computational complexity of the method increases with the number of parts used. The authors reported an average processing rate of 30 fps while using five parts.

4 Proposed DS-KCF tracker

In this section, we provide a detailed description of the core modules comprising DS-KCF, which extend the KCF tracker in a number of different ways. We integrate an efficient combination of colour and depth features in the KCF tracking scheme. We provide a change of scale module, based on depth distribution analysis, that allows to efficiently modify the tracker's model in the Fourier domain. Different from other works that deal with change of scale within the KCF framework, such as [11, 31], our approach estimates the change of scale with minimal impact on real-time performance. We also introduce an occlusion handling module that is able to identify sudden changes in the target region's depth histogram and to recover lost tracks. Finally, a change of shape module, based on the temporal evolution of the segmented target's silhouette, is integrated into the framework. To improve the tracking performance during occlusions, we have added a simple Kalman filter motion model.

A detailed overview of the modules of the proposed tracker is shown in Fig. 2. Initially, depth data in the target region are segmented to extract relevant features for the target's depth distribution (Sect. 4.1). Then, modelled as a Gaussian distribution, changes in scale guide the update in the target's model (Sect. 4.2). At the same time, region depth distribution is deployed to enable the detection of possible occlusions. During an occlusion, the model is not updated and the occluding object is tracked to guide the target search space (Sect. 4.3). Kalman filtering is used to predict the position of the target and the occluding object in order to improve the occlusion recovery strategy. Further, segmentation masks are accumulated over time and used to detect significant changes of shape of the object (Sect. 4.4).

We note that tracking both the occluding and the occluded objects simultaneously during occlusion bears resemblance to multi-object tracking [29]. However, DS-KCF tracks the occluding object to better predict the search area for the target object. After occlusion, the occluding object is not tracked further and its learnt model is not maintained.

4.1 Fast depth segmentation

The first improvement on the KCF tracker we introduce is to exploit depth information to add robustness to its RGB-based approach. Indeed, scale changes, occlusion handling, and shape changes are all made possible by the proposed two-stage depth segmentation approach: (a) a fast one-

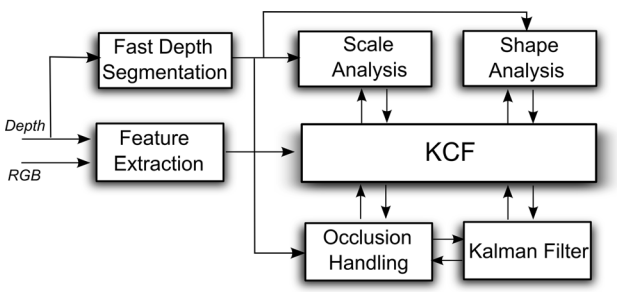

Fig. 2 Block diagram of the proposed DS-KCF tracker

dimensional (1D) implementation of K-means to estimate initial clusters or regions of interest (ROI), followed by, (b) connected component analysis that incorporates spatial features to refine the ROI.

The computational burden of K-means [34] depends on the number of clusters, the number of points to be clustered, and the dimensionality of the features. Moreover, its main drawbacks are that the number of clusters K has to be known a priori and that its convergence is very sensitive to the initial cluster centroid ($\mathbf{d}^{t_0}$). We reduce the effect of this problem by applying K-means to a tracked region's 1D depth histogram and hence reduce the number of features and the number of points to be clustered. A similar approach was proposed for colour image segmentation in [8]. We initialize K and $\mathbf{d}^{t_0}$ with the number and the corresponding depth values of the local maxima of the equally spaced depth histogram, respectively. The estimated $\mathbf{d}^{t_0}$ are then good initial seeds for the K-means algorithm that impose a reduction in its convergence time. The depth histogram $h(d_j)$ is composed of j bins with depth values d_j. Each bin is assigned to the closest cluster k in the depth space. Finally, and until the algorithm stops upon convergence, the cluster centroids are updated with

$$d_k^{t+1} = \frac{\sum_{d_j \in k} d_j \cdot h(d_j)}{\sum_{d_j \in k} h(d_j)}. \tag{9}$$

In the second stage, connected components are formed from the K-means output clusters in the image plane to distinguish between objects located within the same depth plane, and to remove clusters corresponding to small regions. The target region Ω_{obj} corresponds to the cluster with the minimum mean depth. The cluster's mean μ_{obj} and standard deviation σ_{obj} are constantly re-estimated while the object is tracked. Note that the target region, Ω_{obj}, will be exploited further for managing shape changes.

The histogram $h(.)$ bin width can influence the results of the segmentation. This is selected adaptively according to the tracked object's standard deviation (σ_{obj}) as well as the noise model of the depth device [5, 25]. By characterizing the segmented depth data with d_{obj} and σ_{obj}, we use it to deal with the enhancements we propose.

The main advantage of the proposed approach is the trade-off between segmentation accuracy and low computational requirements. As will be shown in more detail in Sect. 5, our depth segmentation has a minor impact on the overall processing time with respect to the core KCF-based tracking.

4.2 Detecting and handling scale changes

As described in Sect. 3, one way that the KCF tracker achieves faster throughput which is by substituting convolution in the spatial domain with element-wise multiplication in the frequency domain. However, element-wise operations expect the matrices to be of the same size. Accordingly, we propose to estimate the target object scale s_{obj} by scaling the object's template relative to its initial depth $d_{\text{obj}}^{t_0}$.

Our approach utilizes two types of scale factors: the first is a continuous scale factor $\mathcal{S}^r = d_{\text{obj}}/d_{\text{obj}}^{t_0}$ obtained from the relative depth of the target, and the second is a set of quantized scale factors $\mathcal{S}^q = \{s_j, (\forall j = 1, \ldots, J)\}$, which enables precomputing different matrices (the regression targets of the training phase and the preprocessing cosine windows in Sect. 3). The current scale is chosen to be the closest level $s_j \in \mathcal{S}^q$ to $\mathcal{S}^r$. We use $\mathcal{S}^q$ to improve computational efficiency and tracker robustness as the models and coefficients are not constantly re-estimated and at the same time $\mathcal{S}^r$ helps to refine the tracker's output as it represents a finer change in scale.

The change of scale detection mechanism is integrated in the KCF processing pipeline as follows. Once the new target position has been estimated, we apply our two-step depth segmentation algorithm and estimate the depth of the target object d_{obj}, $\mathcal{S}^r$ and the corresponding level $s_j \in \mathcal{S}^q$. When a change in scale level s_j is detected, the model template needs to be updated in the Fourier domain and we use *interpolation* and *decimation* for increase and decrease in scale, respectively.

This solution is more stable than model reinitialization, as proposed for example in [47], especially in case of very frequent changes of scale, as the object appearance history is continuously deleted, leading to drift. The proposed approach allows us to preserve this information. Model resampling overhead is added when the tracker moves to a different scale level in $\mathcal{S}^q$. Yet, during tracking, the proposed method guarantees that only one target model at scale s_{obj}, corresponding to the selected s_j, is kept and updated. In such cases, we avoid maintaining and searching a pool of scale templates as many other methods propose, such as [11, 13, 31, 35, 48], due to the fact that the change of scale is guided by the more reliable and fast information provided by the depth segmentation.

The proposed depth-based scale estimation system is valid only under the assumption that the focal length of the camera does not change. On that basis, the bounding box size in the image plane and the actual size of an object have a linear relationship that depends on the depth of the object in the scene. Consequently, the estimated $\mathcal{S}^r$ indicates a real change of scale. Should the camera focal length change be known, the same approach can be applied, the new focal length can be used as a correction factor, and the value of $d_{\text{obj}}^{t_0}$ can be properly scaled.

Resizing in the Fourier domain When the target size is increasing, it is necessary to upsample the model in the Fourier domain. In scaling up, interpolation involves zero padding the higher frequency Fourier coefficients to fit the increased template size, and then adjusting the frequency component amplitude. This is due to the duality between the spatial and Fourier domain and can be easily shown in the following example.

Let us for simplicity start our analysis in the spatial domain and consider a 1D signal $f(n)$, for which we want to increase the number of samples by a factor M. *Interpolation* inserts $M-1$ zero samples such that the new set of samples $g(nM+m)$ is equal to $f(n)$ when $m=0$ and zero otherwise. Now, we can express the DFT of $g(nM+m)$ as

$$\begin{aligned} G(k) &= \frac{1}{\sqrt{MN}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} g(nM+m) e^{\left(\frac{-j2\pi k(nM+m)}{MN}\right)} \\ &= \frac{1}{\sqrt{MN}} \sum_{n=0}^{N-1} g(nM) e^{(-j2\pi k(nM)/MN)} \\ &= \frac{1}{\sqrt{MN}} \sum_{n=0}^{N-1} f(n) e^{(-j2\pi k(nM)/MN)} = \frac{F(k)}{\sqrt{M}}, \end{aligned} \tag{10}$$

which shows how the Fourier coefficients $G(k)$ of the upsampled signal g can be calculated starting from the coefficients $F(k)$ of the original signal, for the scaling factor M. Importantly in DS-KCF, the zero Fourier coefficients are substituted by the corresponding ones from the new patch.

When the target is reducing in size, *decimation* retains the lower portion of the coefficients and discards those of the highest frequencies. In fact, while increasing the spatial sampling in the image, the frequencies spanned by the DFT are reduced [16].

To gain an intuition for the efficacy of this approach, consider upsampling and downsampling an image in the Fourier domain. When upsampling, higher-resolution spatial features are interpolated—avoiding a pixelation in the upsampled image. Conversely, when downsampling, the coefficients representing lower frequencies are retained as the image is now at a coarser resolution. In both cases, previous models' information can be partially preserved and used to build a robust tracker.

In our case, rather than zero padding the previous model's Fourier representation we instead pad it with the coefficients from the current frame and adjust the Fourier gain accordingly—specifically it is set to one.

Scale change parameter selection: accuracy versus processing speed To analyse how the proposed multi-scale handling module affects the performance of DS-KCF tracker, we use the *zcup_move_1* sequence from the Princeton RGB-D dataset [39] as it contains smooth, decreasing scale changes up to 0.5 of the initial target size. We then reverse the frame in order to obtain a sequence with an increasing scale change up to twice the initial object size.

If the scale-steps represented in $\mathcal{S}^q$ are small, the model will be regularly updated and the scale of the model will closely match that of the tracked region being considered. This has a computational overhead as the model will be regularly interpolated and decimated in the Fourier domain. Conversely, if the scale-steps are relatively large, the situation will arise where the model is either too small or too large.

If the model is too small, the tracked region often drifts within the margin of the tracked object, thus reducing precision. If it is too large, the background is incorporated into the model and the tracked region is prone to remain 'stuck' to background when the target moves. Note that small discrepancies in scale have minimal impact on precision.

Figure 3 shows the area under curve (AUC) measure for the forward and reverse *zcup_move_1* sequences against different *scale-steps*, i.e. the difference between each pair of consecutive values, in $\mathcal{S}^q$. The grey line in Fig. 3 shows results for when the tracker's output is computed using the quantized size in $\mathcal{S}^q$ and the blue line when the tracker's output is estimated using $\mathcal{S}^r$. The dotted purple line represents the AUC values for the KCF tracker. The values have been normalized with respect to the maximum, and the black line is the normalized fps processing rate.

It is apparent from this example that the *scale-steps* range of values between 0.08 and 0.2 provide a good trade-off between performance and processing rate. Note though, that compared to the KCF, the proposed DS-KCF obtains better results even for very wide *scale-step*.

For both test sequences, the use of $\mathcal{S}^r$ improves the performance of the tracker especially for larger *scale-step*. However, $\mathcal{S}^r$, considered in isolation, does not reduce the error in precision, which generally remains unchanged, but can be very high, especially in those cases where the scale of the object increases. A qualitative example is shown in Fig. 4b where only a portion of the object is tracked (the inner red bounding box), due to a mismatch of the real object size and current $\mathcal{S}^q$ value. The overlap between the tracked object and the real one is increased if the output of the tracker is corrected by using $\mathcal{S}^r$ (the green bounding box), improving the performance when considering the success plot. However, the position error remains the same, as the tracker is still centred in the same position. When the *scale-step* selected is properly chosen, the object is correctly tracked (see Fig. 4c).

In Fig. 5, we illustrate how an erroneous *scale-step* results in an inferior precision curve for the reversed *zcup_move_1* sequence. Finally, Fig. 6 shows another example of how managing scale changes affects the

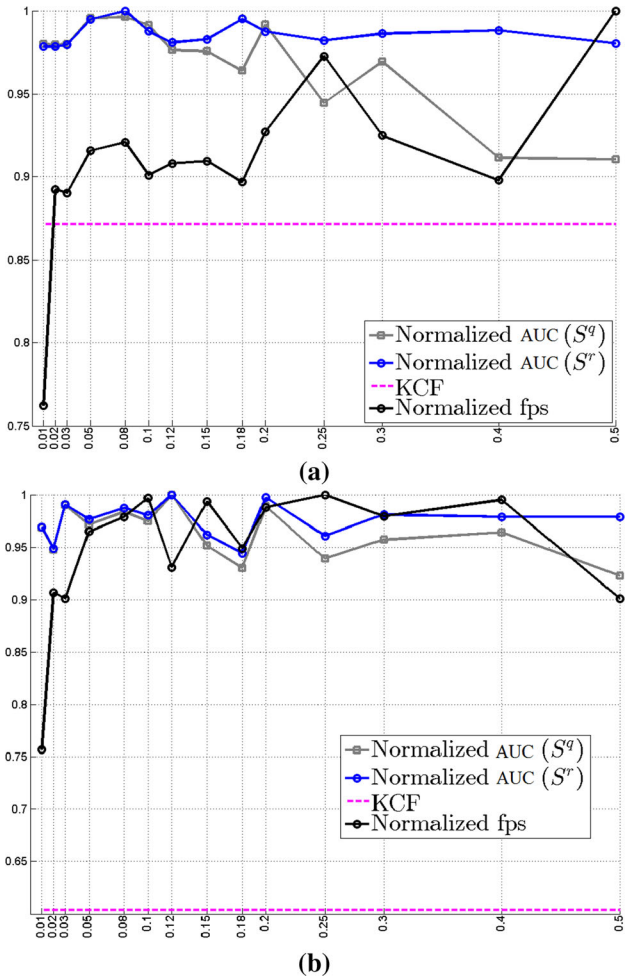

Fig. 3 *Scale-step* analysis: **a** *zcup_move_1* sequence, **b** reversed *zcup_move_1* sequence. Normalized AUC obtained by using the quantized size in $\mathcal{S}^q$ and $\mathcal{S}^r$, in *grey and blue*, respectively; normalized processing rate (fps) in *black*. *Dashed purple line* is the normalized AUC for the KCF tracker. The *x* axis represents the *scale-step* used

processing throughput of the proposed tracker. The overlap score (between the target groundtruth bounding box and the tracked object bounding box) for each frame of the *zcup_move_1* sequence is reported with blue markers, and the normalized processing time is reported with black markers. The positions of the changes of scale are marked with a red circle. Note that when a change of scale is

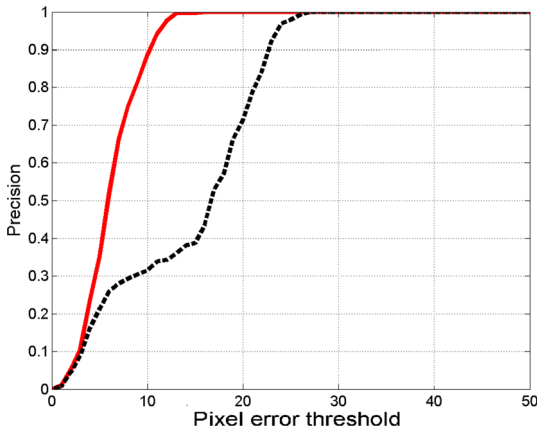

Fig. 5 Precision plot for the reversed *zcup_move_1* sequence for different *scale-step* values 0.1 (*red*) and 0.4 (*black*)

detected, the processing time of the corresponding frame drops significantly, and that the processing rate increases in correspondence to the target scale reduction towards the end of the sequence. The overlap score for the KCF tracker (magenta markers) shows that DS-KCF outperforms it, with particular loss of accuracy in the last part of the sequence where the change of scale is present.

In all our experiments, we used *scale-step* = 0.1. Furthermore, the estimated $\mathcal{S}^r$ was used to further improve the tracker performance.

4.3 Detecting and handling occlusions

We model the tracked object's depth distribution as a single Gaussian and identify candidate occluding objects as the regions not belonging to this model. The work proposed in [39] based on similar concepts as a depth cue is used to discriminate the tracked object from an occluded one. However, our approach differs from [39] in that it is optimized to keep and exploit the advantages of KCF tracking core. In particular, we introduce local search for target candidates by considering depth continuity between the occluded target and the candidates. The occluding regions are segmented with our fast depth segmentation method described earlier and are tracked with the KCF. We use a linear, constant velocity model, Kalman filter to track the position of both the occluded target and the occluding

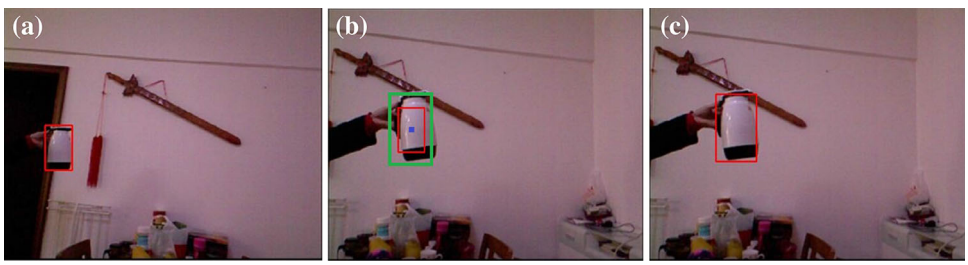

Fig. 4 Example of object scale change: **a** initial tracked object, **b** object wrongly tracked with *scale-step* = 0.4, **c** object correctly tracked with *scale-step* = 0.1

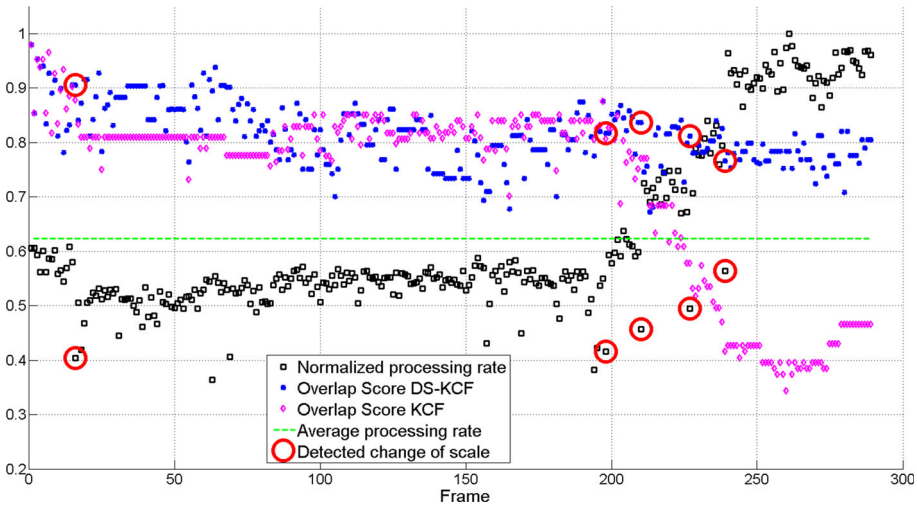

Fig. 6 Change of scale effect (*zcup_move_1* sequence). Overlap score: *blue markers* for DS-KCF and *magenta markers* for KCF, normalized processing time *black square markers*, detected changes of scale are enclosed in the *red circle*. Average processing rate is reported in *green*

object during occlusions (see later for details). The state variables of the Kalman filter are the position and velocity of the bounding box centroids of both objects. The use of the 2.5D data (image coordinates with real-world depth) facilitates optimization of the search space [see Eq. (12) later] and yields a more accurate estimate of the position of the occluded target. This is another improvement on our earlier version of DS-KCF [6] which boosts the tracking performance as will be shown in Sect. 5.

Occlusion detection We define μ_{obj} as the mean depth of the tracked object and σ_{obj} as its standard deviation. Given a segmented region Ω_{obj}, the corresponding rectangular ROI Γ_{obj}, and its corresponding patch $\mathbf{z}$ used by the core KCF tracker, then occlusion is detected if

$$\left(\Phi\left(\Gamma_{\text{obj}}\right) > \lambda_{\text{occ}}\right) \wedge \left(\widehat{f(\mathbf{z})}_{\max} < \lambda_{r1}\right), \tag{11}$$

where $\Phi(\Gamma_{\text{obj}})$ is the fraction of pixels belonging to Γ_{obj} up to two standard deviations from the object's mean. We have determined empirically that an occlusion should be detected when approximately a third of Γ_{obj} is occupied by the occluding object, i.e. $\lambda_{\text{occ}} = 35\%$. The second term in Eq. (11) reduces false detections of occlusion in the case of objects moving quickly towards the camera. In such situations, the overlap condition can be satisfied due to a fast shift in the object's depth distribution. However, the maximum response of our tracker, $\widehat{f(\mathbf{z})}_{\max}$, would still be high, with $\widehat{f(\mathbf{z})}_{\max}$ obtained by weighting $f(z)$ in Eq. (7) with a sigmoidal function that takes into account the distance between the depth data and μ_{obj} to guarantee continuity on depth. The value of λ_{r1} was also determined empirically as $\lambda_{r1} = 0.4$.

When Eq. (11) is not satisfied, the tracking process progresses normally and the model is updated according to the current scale and the tracker is ready to process the next frame. Conversely, when Eq. (11) is true, the tracker enters in the occlusion state and a new KCF tracker is initialized for tracking the occluding object. Note, only a portion of the occluding object is contained in Γ_{obj}. To obtain the entire occluding object and the depth values of μ_{occ} and σ_{occ}, the connected component is extracted from the depth frame.

Occlusion state In a state of occlusion, a search region Γ^i_{search} at frame i is defined, and its response is computed to detect the reappearance of the target. The target object is highly likely to re-emerge in those image areas gradually uncovered by Γ_{obj}.

We identify the region where target candidates are searched as

$$\Gamma^i_{\text{search}} = \Gamma^{i-1}_{\text{occ}} \cup \Gamma^{i-1}_{\text{bc}} \cup \Gamma^i_{\text{occ}} \cup \Gamma^i_{\text{KF}}, \tag{12}$$

where Γ^{i-1}_{bc} is the region previously occupied by the *b*est target *c*andidate (Fig. 7). For tracking the occluding object, the accuracy of the KCF tracker is sufficient for our purposes as our goal is to only have a rough estimate of Γ_{occ}. The last term, Γ_{KF}, represents the region corresponding to the estimated target position provided by the Kalman filter.

For each cluster with depth mean μ_n and position c_n in the image plane, we compute the maximum value of the normalized response $\widehat{f(z)}_n$. The best target candidate corresponds to the one with maximum response $\widehat{f(z)}_n$. Target tracking is resumed when

Fig. 7 Occlusion search region: Γ^i_{search} (*blue line*), Γ^i_{occ} (*yellow line*), Γ^i_{obj} (*red line*), Γ_{KF} (*green line*)

$$(\Phi(\Gamma_{\text{bc}}) < \lambda_{\text{occ}}) \wedge \left(\widehat{f(z)}_n > \lambda_{r2}\right), \tag{13}$$

with $\lambda_{r2} = 0.2$ empirically determined.

The computational requirements of DS-KCF could be significantly affected during occlusion if the entire Γ_{bc} is scanned. We reduce the computational requirements by enabling the response estimation only in specific key areas, i.e. the clusters in Γ_{search}. The search for the best candidate can be further optimized by parallelizing the estimation of the response in each cluster, allowing us to reduce the impact of occlusions on the overall processing rate even further.

Kalman filtering of the target An important contribution to the definition of the search area Γ_{search} in Eq. 12 comes from Γ_{KF}. Assuming the target will move at constant velocity, Γ_{KF} represents a good estimate of the target candidate region by the Kalman filter, and this is particularly helpful during occlusions. Applying the Kalman filter to the 2.5D coordinate of the target's centroid, the state of the target is modelled as $\mathbf{p} = u, v, d, \dot{u}, \dot{v}, \dot{d}$, where u and v are the centroid location, n is the image plane, and d is the depth mean value of the tracked object. The system state and observation equations are

$$\begin{aligned}\mathbf{p}_t &= \mathbf{A}\mathbf{s}_{t-1} + \eta_{t-1} \\ &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{t-1} + \eta_{t-1},\end{aligned} \tag{14}$$

$$\begin{aligned}\mathbf{o}_t &= \mathbf{H}\mathbf{p}_t + v_t \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{p}_t + v_t,\end{aligned} \tag{15}$$

where $\mathbf{o}$ is the observation vector obtained with the DS-KCF (target position and d estimated as in Sect. 4.1). $\mathbf{A}$ and $\mathbf{H}$ are, respectively, the constant velocity model state transition matrix and the measurements matrix, respectively, and η and v represent model uncertainty and noise of the measurements. In summary, during normal tracking the DS-KCF output is used to update the Kalman filter system state. If the DS-KCF enters into occluded state, the Kalman filter state is used to predict the position of the target (Γ_{KF} in Eq. 12).

4.4 Detecting and handling shape changes

As in previous works in object tracking, our aim is to find the smallest bounding box that encapsulates the tracked object. While Sect. 4.2 deals with changes in scale, the bounding box is likely to change its aspect ratio due to in-plane or out-of-plane rotations for rigid objects or due to changes in shape for deformable objects.

The 2D projection of any 3D object's bounding box may be faithfully represented by combining a change in scale with a change in aspect ratio. Note that for some elongated objects that are rotated at an oblique angle in the image plane, this representation is less useful as the majority of the region defining the target contains background information. However, in most situations this representation offers clear advantages over the usual paradigm where bounding box aspect ratio is fixed, for example in human pose transitions such as lying to sitting to standing. Similarly, when an object is occluded, the aspect ratio of the tracked bounding box should change so it is confined to the unoccluded part of the object. This section accordingly enables tracking the change in aspect shape of the tracked bounding box, as a result of shape change, during occlusion and after the object emerges from behind an occluder. To the best of the authors' knowledge, the proposed approach is the first to propose a solution to change of shape handling in the KCF framework.

One of the major strengths of KCF and DS-KCF is that they operate solely in the frequency domain. While scaling in DS-KCF is achieved via interpolation and decimation, altering the aspect ratio of the tracked patch in the Fourier domain is problematic as spatial information is encoded in phase. The main idea of the proposed change of shape module is to maintain the advantages of the frequency domain operation at the core of KCF. Specifically, we wish to avoid the computational load that would be incurred to reinitialize or adapt the target model in the spatial domain every time there is a significant change in shape. Hence, we fix the aspect ratio of the tracked patch, and the target model, and subsequently adjust the aspect ratio of the new shape of the object.

We tried two approaches for selecting a tracking patch with a fixed aspect ratio (rectangular or square) The first approach uses the aspect ratio extracted from the initial rectangular template and keeps it rectangular while the template is updated as the object is segmented and tracked. The second approach is based on a square patch (i.e. aspect ratio is maintained as 1) centred on the target region, which is scaled such that the edge length always matches the shortest spatial extent of the target object in the image plane. This square is then scaled such that it fits the depth-based segmentation. After the patch is tracked, the spatial extent of the target object is adjusted based on the depth segmentation to encapsulate the full extent of the object. To avoid drifts due to segmentation failure, a temporal window is considered. The union of the segmented regions, Ω_{obj}, from the last n frames is used,

$$\Omega_{\text{shape}} = \bigcup_{t=1}^{n} \Omega_{\text{obj}}^{t}. \tag{16}$$

since it produces better results than the average.

As in Sect. 4.3, we use the symbol Γ to indicate the region corresponding to the segmented area Ω and the tracked target area. The reported bounding box Γ_{output} is computed such that

$$\Gamma_{\text{output}}^{t} = \begin{cases} \Gamma_{\text{shape}}^{t} & \dfrac{\Gamma_{\text{shape}}^{t}}{\Gamma_{\text{output}}^{t-1}} < 0.1 \\ \Gamma_{\text{output}}^{t-1} & \text{otherwise} \end{cases} \tag{17}$$

The threshold of 10% increase is used to maintain smooth shape transitions and ensure the depth model, μ_{obj} and σ_{obj}, does not drift into other depth regions. An example is shown in Fig. 8.

When using a square patch, if an object is occluded, and the tracked square patch remains unoccluded, the occlusion state would not be detected. This affects the estimation of the object's depth model, μ_{obj} and σ_{obj}. Failing to detection occlusions would also affect the occlusion recovery [Eq. (13)]. Moreover, the tracker's maximum response $\widehat{f(\mathbf{z})}_{\max}$ would not correspond to the object's centroid which is required for measuring the tracker's localization performance. For this, we use the centroid of the segmentation area, Γ_{output}, to indicate the object's current position. However, results show that the square patch produces less accurate tracking in cases of partial occlusions. On the other hand, when a rectangular patch of fixed aspect ratio is used, the tracker's response to major occlusions would be adversely affected. For these reasons, we use the rectangular template in our shape handling module.

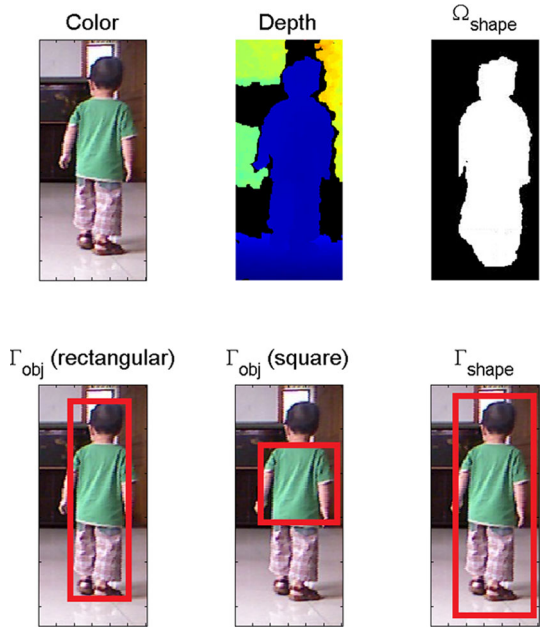

Fig. 8 Example of Ω_{shape}, Γ_{obj} for the *rectangular* and *square* templates, and Γ_{shape}

Handling shape changes: pros and cons We illustrate the advantages of being able to manage shape changes using two sequences, (a) *child_no1* from [39] which shows a child squatting to play, and thus, the target's aspect ratio changes and (b) the *milk* sequence from [15] which depicts pouring from a milk box and thus has an in-plane rotation of a rigid object.

In Fig. 9, qualitative results for DS-KCF with and without shape handling are reported. The first column contains the initial target, while the second column shows the object after the change of shape. DS-KCF with shape handling is able to track the object more accurately. In the *milk* example, even though shape handling improves the results, it does not fully segment the object due to the constraint on smooth shape changes in Eq. (17).

In Fig. 10, the overlap scores for DS-KCF with and without shape handling are reported for the two sequences. For the *child_no1* sequence (Fig. 10a), the graph shows that the accuracy is comparable when the shape of the object remains constant. However, when a change of shape takes place (within the two vertical dotted lines), better accuracy is obtained through shape change handling. The analysis of the performance on the *milk* sequence shows both the advantages and drawbacks of shape handling. During changes of shape, it is evident that the shape handling functionality produces superior results. However, the accuracy drops later as the hand and milk box have similar depths, resulting in a combined segmentation. The same occurs when the milk box is repositioned on the table. As

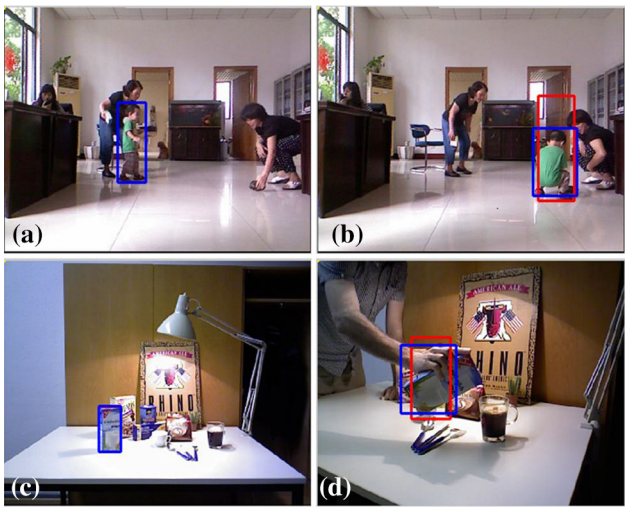

Fig. 9 Change of shape examples for sequences *child_no1* (**a**, **b**) and *milk* (**c**, **d**). DS-KCF with shape handling are in *blue*, and without in *red*

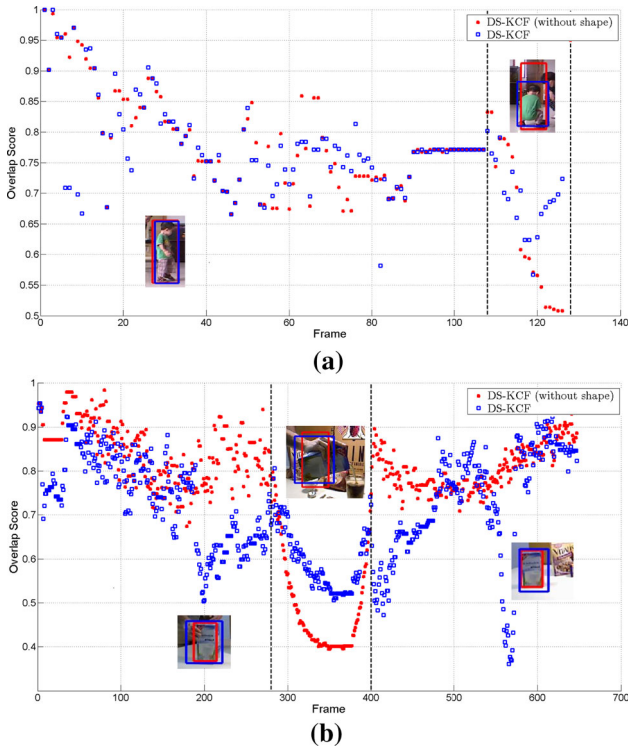

Fig. 10 Change of shape effect for *child_no1* sequence (**a**) and *milk* (**b**). Overlap score for DS-KCF with shape handling is in *blue* and without in *red*. Strong changes of scale are enclosed between *black vertical lines*

previously discussed, there is a trade-off between segmentation and the tracking accuracy.

Change of shape module integration The change of shape module exploits the information extracted within the other DS-KCF modules with minimal impact on the system throughput (as evaluated in Sect. 5). Effectively, this module takes advantage of the estimated target depth distribution and segmentation mask (see Sect. 4.1) that is accumulated in Eq. 16. Thus, it is possible to capture the shape dynamics and to identify a change of shape when Eq. 17 is satisfied. The detected changes of shape modify only the tracker's output, while its model is updated normally based on the current scale.

5 Experimental results

We now provide an in-depth evaluation of the proposed tracker's performance using the Princeton dataset [39] which was recorded with the Microsoft Kinect. It comprises 100 videos containing both RGB and depth data. The manual annotated groundtruth is available for five videos that form the validation set and the outcome for the test set can be obtained by submitting the tracker's output online[1] on the comparison site set up by author's of [39]. It must be noted that we discovered synchronization errors between the RGB and depth data for all of the videos in the Princeton dataset [39]. These can affect the stability of RGB-D trackers and the validity of the results as the groundtruth is based solely on the RGB frames. In the light of this, we realigned the data streams, and consequently the groundtruth, for the five validation sequences. The videos depict indoor scenes with object depth values ranging from 0.5 to 10 m. Different targets are present in the scene, e.g. humans, animals, and rigid objects. Each scene presents a different level of background clutter, ranging from simple and static backgrounds to more complex scenes, for example a university hall. Additionally, there are sequences with people moving around their environment with intermittent occlusions.

We evaluate the performance of the tracker by considering two widely used scores, precision and success plots (see [43]). Precision plots are obtained by computing the percentage of frames for which the location error is below a certain threshold. As the representative precision score, we use the value obtained for the threshold as 20 pixels (P20), as proposed in [43]. Success plots measure the bounding box overlap between the tracked object and the groundtruth and provide the percentage of successful frames where the overlap is larger than a threshold as it is varied from 0 to 1. For these, we report the area under curve (AUC). We also provide the computational performance of the methods in terms of processed frames per second.

We compare the proposed approach with different state-of-the-art RGB and RGB-D trackers that have been tested on the Princeton dataset [39]. All our experiments (for both MATLAB and C++ code) have been carried out on a workstation with a quad-core Intel I7-3770S 3.10GHz CPU with 8GB of RAM.

5.1 Feature analysis

The goal of this section is to analyse the performance of the proposed tracker when using different RGB and depth feature combinations and show that the use of depth data is fundamental to boost the tracking performance. As previously mentioned in Sect. 3, both raw colour and HoG features were deployed in the original KCF [22] where it was demonstrated that HoG features provide higher accuracy without compromising real-time performance. Hence, we analysed five different HoG-based feature combinations: colour data (*hog_colour*), depth (*hog_depth*), two independent models based on colour and depth and obtaining the overall response as the their linear combination (*hog_linear*), using the same two models but selecting the maximum response (*hog_max*), and finally combining colour and depth for a concatenated feature

[1] http://tracking.cs.princeton.edu/eval.php.

representation (*hog_concat*). Furthermore, we tested the proposed method with state-of-the-art RGB-D features (named *gPb+D*) for object segmentation and object recognition introduced in [17, 18]. These features combine local geometric gradients, i.e. depth gradients, convex and concave normal gradients, and colour-based gradients of brightness, colour, and texture. Even if the computational cost of *gPb+D* features is not sustainable for real-time tracking scenarios, it is important to verify their impact on the tracker's accuracy. For the sake of simplicity, we report the results for the DS-KCF tracker without using the change of shape handling module as the validation set does not contain sequences with significant shape variations.

Precision and success plots computed on the Princeton validation set are shown in Fig. 11. The curves show that *hog_colour*, *hog_max*, and *hog_concat* give very similar results, while *hog_linear* yields the worst results. AUC, P20, and fps measures for these curves (obtained with MATLAB) are reported in Table 2 with the best results indicated in bold text. KCF clearly is very fast but has significantly poorer average performance. The best average performance for both AUC and P20 measures (at 79.5% and 94.2%, respectively) is obtained when using *hog_concat*, at a respectable 40 fps. The state-of-the-art *gPb+D* features have a considerably lower performance at a much lower frame rate, even if they are computed in small image patches.

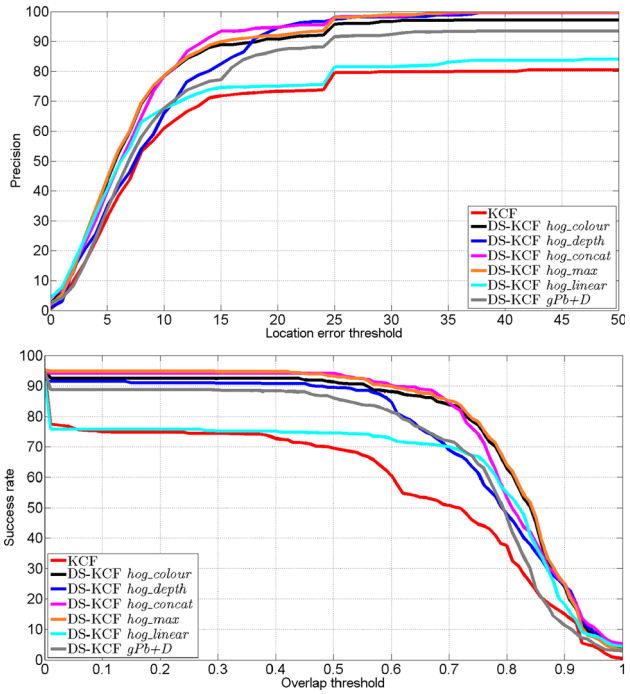

Fig. 11 Different features and importance of depth—KCF versus DS-KCF: (*top*) average precision plot and (*bottom*) average success plot

In the rest of the experiments, *hog_concat* was used as the feature of choice by the proposed DS-KCF tracker.

5.2 Real-time performance and implementation details

We provide a detailed analysis of the real-time performance of the proposed DS-KCF and its main modules by considering the different scenarios contained in the 100 sequences of the Princeton dataset. We present the results of the MATLAB implementation of the DS-KCF with and without the new shape handling module and the C++ implementation of the DS-KCF (without shape handling).

Table 3 shows the average processing rate for each of the five sequences in the Princeton validation set, and the percentage[2] or processing load of the total time spent by each fundamental module of the system. The modules considered were: *MR*—evaluation of the maximum response relative to operations in Eq. (7), *Seg+Occ*—target patch segmentation, as in Sect. 4.1, and occlusion estimation as in Eq. (11), *TRocc*—occluder tracking, *OccSolve*—occlusion solving as in Eq. 12, *SE*—change of scale estimation as in Sect. 4.2, and finally *MU*—the model update as presented in Sect. 3.

One main advantage of the proposed approach is that change of scale estimation is optimized, thanks to the use of depth, without analysing any search space. As discussed in Sect. 4.2, maintaining and searching a pool of scale templates, as many other methods propose, such as [11, 13, 31, 35, 48], require a higher computational cost since the convolution operations between the different scale templates and the tracked patch has to be repeated for all the scales (or a subset of it). The benefits of this aspect of our approach are very clear if we consider the SE column in Table 3 where for all the proposed trackers the *SE* processing load is lower than 0.5%. In the case of sequences without occlusions (*child_no1* and *zcup_move_1*), the modules that have the highest processing load are *MR* and *MU*, while the proposed depth segmentation has generally very little impact, especially for our C++ implementation, and provides a good trade-off between accuracy and processing speed without affecting the KCF tracking core speed.

In the sequences that include occlusions, we see that modules *TRocc* and *OccSolve* affect the tracker's performance considerably. During occlusion, we track the occluding object and we need to test the response of the tracker in different candidate regions. Even if the candidate regions are efficiently selected, we are repeating the same very demanding operations in *MR* several times. With the

[2] The percentages in each row may not sum to 100 due to other processing overheads.

Table 2 Performance of the DS-KCF with different features on the Princeton dataset's [39] validation set

	Validation set [39]																	
	bear_front			*child_no1*			*zcup_move_1*			*face_occ5*			*new_ex_occ4*			*Average*		
	AUC	P20	fps	AUC	P20	fps	AUC	P20	fps	AUC	P20	fps	AUC	P20	fps	AUC	P20	fps
KCF [22]	18.6	19.8	**117**	66.3	**96.8**	**55**	72.5	**100.0**	**164**	79.7	93.1	**88**	46.1	56.8	**92**	56.6	73.3	**103**
DS-KCF *hog_colour*	73.6	82.4	64	**83.2**	93.7	33	82.3	**100.0**	105	69.9	80.7	38	78.1	90.2	61	77.4	89.4	60
DS-KCF *hog_depth*	**81.9**	**91.3**	54	69.0	91.3	34	78.8	**100.0**	111	82.2	95.1	51	53.5	58.5	33	73.1	87.2	57
DS-KCF *hog_concat*	75.7	85.3	42	76.5	92.1	19	81.5	**100.0**	81	**85.5**	**98.6**	22	78.3	95.1	38	**79.5**	**94.2**	40
DS-KCF *hog_max*	72.1	81.3	45	83.1	92.1	20	82.3	**100.0**	74	79.4	92.2	37	78.1	90.2	40	79.0	91.2	43
DS-KCF *hog_linear*	67.7	74.2	35	82.6	92.1	20	**82.5**	**100.0**	75	85.0	**98.6**	26	10.5	9.1	67	65.6	74.8	45
DS-KCF *gPb+D*	49.6	55.2	4	67.5	88.9	2	81.9	**100.0**	8	71.3	91.4	3	**78.7**	**100.0**	7	69.8	87.1	4

All measures are in percentages, except for the fps columns

Table 3 Processing rate results: module processing loads (percentages) and average processing rate (fps) for different videos of the Princeton dataset validation set [39]

Validation set [39]	Algorithm	*MR*	*Seg+Occ*	*TRocc*	*OccSolve*	*SE*	*MU*	Avg. fps
bear_front	**DS-KCF no shape**	15.9	12.3	12.6	38.4	0.1	15.9	45.9
	DS-KCF no shape C++	37.1	4.0	15.2	8.8	0.1	30.3	127.4
	DS-KCF	16.7	26.4	12.3	22.8	0.5	16.8	39.6
child_no1	**DS-KCF no shape**	42.8	11.5	–	–	0.1	44.3	18.9
	DS-KCF no shape C++	47.6	9.8	–	–	0.2	42.0	165.2
	DS-KCF	37.1	23.1	–	–	0.1	38.4	16.2
zcup_move_1	**DS-KCF no shape**	34.7	27.8	–	–	0.2	34.2	83.2
	DS-KCF no shape C++	48.8	6.9	–	–	0.1	43.0	223.3
	DS-KCF	27.3	39.8	–	–	0.2	29.6	61.3
face_occ5	**DS-KCF no shape**	22.8	7.8	19.4	25.6	0.1	23.0	21.8
	DS-KCF no shape C++	37.8	5.0	10.4	13.3	0.0	30.8	206.3
	DS-KCF	18.6	24.3	15.7	21.1	0.1	19.0	17.4
new_ex_occ4	**DS-KCF no shape**	31.8	15.7	11.9	11.4	0.1	27.4	37.3
	DS-KCF no shape C++	37.1	4.5	16.9	5.3	0.2	32.3	89.8
	DS-KCF	24.0	22.7	10.7	16.2	0.1	24.4	34.0

All codes are in MATLAB, unless stated otherwise

proposed C++ implementation, we are able to compute the response $\widehat{f(\mathbf{z})}_{\max}$ in all the different regions in parallel, reducing the impact of this module.

The computational performance of DS-KCF is clearly reduced when change of shape handling is enabled, e.g. see the *Seg+Occ* column in Table 3.

A comparison between the different approaches and implementations is given in Table 4. The Princeton test dataset [39] has been partitioned into different categorizations according to the target characteristics, such as object type, size, speed of movement, presence of occlusions, and motion type. We report minimum, maximum, and average processing rates on the 95 test sequences of the Princeton dataset for some of these different categorizations, i.e. target size, speed of movement, and presence of occlusion. The C++ version of the DS-KCF provides a remarkable average frame rate of 186 fps with a an average minimum value for all the sequences of over 67 fps. This is a very important result that indicates that even in the worst case scenario the provided algorithm operates at frame rates exceeding those available on typical RGB-D devices (around 30 fps for Kinect and Asus Xtion). Furthermore, on average, the MATLAB versions guarantee real-time performance with a processing rate greater than 35 fps and a

Table 4 Processing rate results: minimum, maximum, and average processing rate (fps) for different video categories and the entire validation set of the Princeton dataset [39]

Categories	Property	Processing	DS-KCF no shape	DS-KCF (C++) no shape	DS-KCF
Size	Small	Min	20.6	79.7	17.5
		Max	113.2	408.3	108.3
		Avg	44.4	228.3	35.7
	Large	Min	19.8	47.0	17.7
		Max	113.9	202.2	106.7
		Avg	42.0	113.4	34.9
Movement	Slow	Min	22.2	68.4	19.7
		Max	112.8	328.1	108.7
		Avg	46.6	196.7	37.8
	Fast	Min	18.3	66.9	15.4
		Max	114.1	336.7	106.7
		Avg	40.4	175.0	33.0
Occlusion	Yes	Min	10.7	40.0	10.2
		Max	105.5	298.0	99.7
		Avg	32.6	147.7	28.5
	No	Min	33.5	105.6	27.8
		Max	124.3	379.7	118.8
		Avg	58.5	238.6	44.9
Across 95 sequences		Min	20.3	67.7	17.6
		Max	113.4	332.4	107.7
		Avg	43.5	186.0	35.4

All codes are in MATLAB, unless stated otherwise

minimum processing rate of over 17 fps. As previously mentioned, DS-KCF results in slightly lower processing rate when the change of shape module is activated.

From the occlusion column of Table 4, we can see that, as discussed above, the presence of occlusions slows the proposed DS-KCF. On average, the processing rate is reduced by about 40% compared to sequences without occlusions. However, the achieved performance is still considerable with an average of 147 fps and a minimum of 40 fps in the case of the C++ version. For the MATLAB version, the average processing rate is $\approx$30 fps with a minimum value of $\approx$10 fps. Finally, the size of the tracked object also affects tracking performance, while the speed of the object does not affect the tracker processing rate significantly.

The C++ implementation is based on a publicly available object-oriented KCF tracker implementation [19]. This was extended and specialized to produce the DS-KCF tracker. For this extension, we made use of a number of libraries, including; OpenNI for real-time capturing from consumer RGB-D cameras, OpenCV for optimized real-time image operations, and Intel's Thread Building Blocks (TBBs) for cross-platform parallelization. The use of TBB for parallelization limits our implementation to scale with the number of cores available on the CPU and makes no use of the GPU, except implicitly through some OpenCV operations. TBB was used to improve the performance of the *OccSolve* module. As previously mentioned, in this module the tracker must find the best candidate region from a set of regions, which can be evaluated independently. Evaluating the maximum response of each candidate region inflicts a performance penalty which can be mitigated by spreading the load across all of the machine's available cores. This choice of libraries allowed us to optimize performance without compromising platform independence, so our implementation has been tested on Windows and Linux.

5.3 Results with Princeton dataset

Among other state-of-the-art RGB-D segmentation algorithms, the sDasp algorithm presented in [42] provides the best trade-off between accuracy and processing time. However, it can only process a RGB-D frame (resolution of 640x480) at 2 fps and a typical object patch (for example 235x145 from the validation set of [39]) at $\approx$18 fps with a C++ implementation. This processing rate will constitute a bottleneck for the entire tracking process. Moreover, as shown in Fig. 12 the target segmentation results obtained with the proposed fast segmentation approach (third column) are very similar to the ones obtained by sDasp (fourth column)—even in the occluding scene example in the third row.

We now compare the results of our proposed DS-KCF tracker against 19 other state-of-the-art methods on the

Princeton test dataset [39]. These other RGB-D trackers are: OAPF [36], RGBDOcc+OF [39], RGBD+OF [39], PCdet_flow [39], SAMF+Depth [31], PCdet, and PCflow from [39]. We also compare the proposed approach with the following RGB trackers: KCF [22] (experiments run by us), RGBOF [39], Struck [21], VTD [28], RGB [39], TLD [24], MIL [1], OF [4], CT [46], SemiB [40]. We also include two depth-based trackers: LDPSTRUCK [2], Dhog [39].

Table 5 summarizes the results obtained by all the different algorithms , showing the average AUC obtained for all 95 videos and for each of the different video categorizations. Moreover, we report an average ranking of the algorithms (second column) by considering the individual rankings under the different categorizations. The methods in the table are ordered according to the best performance obtained, i.e. by their average rank. The representation in Table 5 is as proposed in [39]. However, as the number of sequences in each category is different and some videos belong to more than one category, this ranking is not a perfect valid summary of the results. The combined analysis of AUC and *Avg. Rank* would be more appropriate for a detailed look (see below).

Our proposed method is ranked third (in MATLAB), fourth (without shape handling in MATLAB), and fifth (without shape handling in C++). The small difference between the accuracies obtained by the MATLAB and C++ implementations of DS-KCF arises since the two implementation are completely independent and use different libraries for completing all the operations, such as the Fourier transformations, matrix operations, and also the computation of the fast segmentation approach. Only the HoG feature extraction module belongs to the same software library.

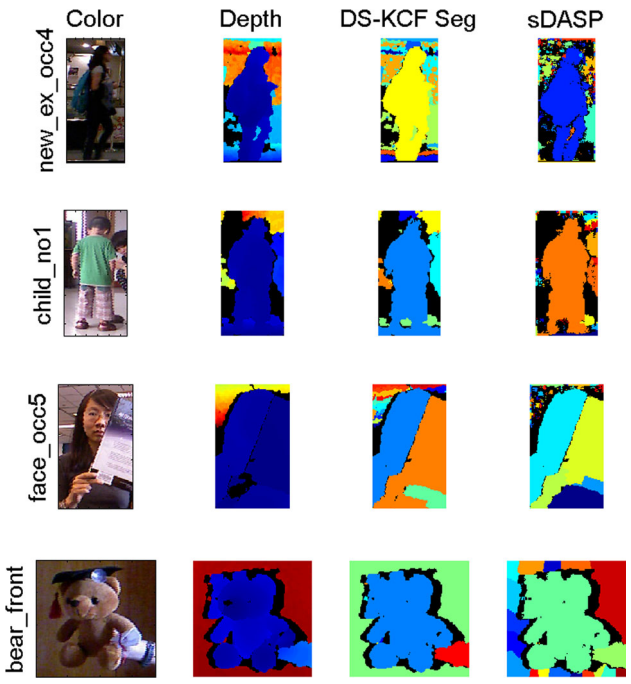

Fig. 12 Example of segmented objects from [39]. Colour and depth data (*first two columns*), proposed segmentation (*third column*), RGB-D segmenter sDasp [42] (*fourth column*)

As may be expected, DS-KCF scores a better *Avg. Rank* and average AUC when with shape handling than when without—for example, it scores an average AUC of 71.9% with shape handling compared to 69.3% otherwise. The shape handling DS-KCF achieves better results for the human and animal categories, as the proposed approach is well suited for the gradual changes of shape exemplified by human and animal movements in the dataset. However, the number of examples in the Princeton test dataset that include shape changes is very limited; hence, our 'with and without shape handling' results are not immensely different. We address this a little later. Also, the system can fail in certain cases where the shape changes are generally fast and the tracked objects are very close to the ground plane (as in the case of the turtle moving in *wuguiTwo_no* sequence in Fig. 13b). Figure 13a shows another example of how a wrong segmentation can lead to a partially incorrect object shape estimate. In this example, the depth-based segmentation includes the hair, which is effectively at the same depth as the tracked object (i.e. the face), hence leading to lower accuracy.

The comparison of occlusion handling performance of the DS-KCF and the no-shape version proposed in our previous work [6] is given in Table 5, where we see a 1.6% improvement in performance with an overall accuracy of 64.9%—that is the second best result for the occlusion category.

Overall, only two approaches, OAPF [36] and RGBDOcc+OF [39], obtain a higher *Avg. Rank* (by $\approx$0.6% more) and a higher average AUC value (by $\approx$1.4% more). However, as reported by their authors, these approaches achieve a very low processing rate of less than 1 fps. Our proposed method has an average processing rate ranging from 35 to 43 fps for its MATLAB implementation and 187 fps for its C++ version.

5.4 More on shape handling

Change of shape events in the Princeton Dataset is rare and, as evident in the examples in Fig. 10, rather limited to a few frames only. This means the efficiency of the shape handling power of DS-KCF requires further verification. To this purpose, we examine DS-KCF on our own special-purpose RGB-D dataset (RotTrack) containing different change of shape scenarios.

RotTrack comprises five RGB-D sequences containing four different types of change of shape as shown in Fig. 14:

Table 5 Evaluation results: AUC and corresponding rankings (in parentheses) under different categorizations

Method	Avg.	Avg.	Target type			Target size		Movement		Occlusion		Motion type	
	Rank	AUC	Human	Animal	Rigid	Large	Small	Slow	Fast	Yes	No	Passive	Active
OAPF [36]	2.3	73.1	64.2 (5)	84.8 (1)	77.2 (2)	72.7 (3)	73.4 (1)	85.1 (1)	68.4 (3)	64.4 (3)	85.1 (1)	77.7 (4)	71.4 (1)
RGBDOcc+OF [39]	2.3	73.3	74.0 (1)	62.6 (5)	78.4 (1)	78.1 (1)	69.7 (3)	76.3 (3)	72.2 (1)	72.0 (1)	75.2 (6)	82.3 (1)	70.0 (2)
DS-KCF	2.9	71.9	70.9 (2)	70.8 (2)	73.6 (6)	73.9 (2)	70.3 (2)	76.2 (4)	70.1 (2)	64.9 (2)	81.4 (2)	77.4 (5)	69.8 (3)
DS-KCF no shape [6]	4.3	69.3	67.0 (3)	61.2 (6)	76.4 (3)	68.8 (5)	69.7 (4)	75.4 (5)	66.9 (4)	63.3 (4)	77.6 (5)	78.8 (3)	65.7 (5)
DS-KCF no shape C++	4.5	68.1	64.5 (4)	64.3 (4)	74.3 (5)	66.3 (6)	69.4 (5)	76.5 (2)	64.7 (6)	60.1 (5)	79.0 (4)	79.6 (2)	63.7 (6)
RGBD+OF [39]	5.0	68.1	63.9 (6)	65.3 (3)	74.5 (4)	71.5 (4)	65.5 (6)	73.4 (7)	65.9 (5)	60.1 (6)	79.0 (3)	74.0 (7)	65.8 (4)
PCdet_flow [39]	7.3	58.9	50.5 (7)	51.6 (8)	72.7 (7)	63.4 (7)	55.5 (8)	73.9 (6)	53.0 (7)	55.0 (7)	64.4 (10)	75.5 (6)	52.7 (7)
SAMF+Depth [31]	9.0	54.0	44.8 (10)	49.6 (10)	67.0 (8)	52.4 (9)	55.2 (9)	65.2 (8)	49.5 (10)	41.1 (10)	71.6 (7)	66.3 (8)	49.3 (10)
KCF [22]	10.3	52.0	41.8 (12)	50.4 (9)	64.9 (9)	48.4 (11)	54.7 (10)	65.0 (9)	46.9 (12)	40.6 (11)	67.7 (9)	64.5 (10)	47.3 (11)
RGBOF [39]	10.3	53.2	47.1 (8)	47.0 (13)	63.6 (10)	47.4 (12)	57.5 (7)	56.7 (13)	51.8 (8)	46.9 (8)	61.9 (14)	63.4 (11)	49.3 (9)
LDPSTRUCK [2]	10.4	51.8	46.2 (9)	59.1 (7)	54.5 (14)	51.6 (10)	52.0 (11)	56.2 (14)	50.1 (9)	39.8 (12)	68.4 (8)	56.4 (12)	50.1 (8)
PCdet [39]	12.2	48.7	40.6 (13)	42.1 (16)	61.7 (11)	55.4 (8)	43.6 (16)	58.5 (10)	44.8 (13)	46.3 (9)	52.0 (16)	64.9 (9)	42.6 (13)
Dhog [39]	12.5	49.0	43.3 (11)	48.3 (12)	55.9 (12)	47.2 (13)	50.3 (12)	52.7 (15)	47.5 (11)	38.4 (13)	63.5 (11)	54.3 (16)	46.9 (12)
Struck [21]	14.2	44.4	35.4 (14)	47.0 (14)	53.4 (16)	45.0 (14)	43.9 (15)	58.0 (11)	39.0 (14)	30.4 (17)	63.5 (12)	54.4 (15)	40.6 (14)
VTD [28]	14.6	43.0	30.9 (18)	48.8 (11)	53.9 (15)	38.6 (17)	46.2 (13)	57.3 (12)	37.2 (15)	28.3 (18)	63.1 (13)	54.9 (14)	38.5 (15)
RGB [39]	16.4	39.9	26.7 (20)	40.9 (17)	54.7 (13)	31.9 (20)	46.0 (14)	50.5 (17)	35.7 (16)	34.8 (14)	46.8 (18)	56.2 (13)	33.7 (18)
CT [46]	17.6	36.4	31.1 (17)	46.7 (15)	36.9 (20)	39.0 (16)	34.4 (19)	48.6 (18)	31.5 (18)	23.3 (21)	54.3 (15)	42.1 (18)	34.2 (17)
PCflow [39]	17.8	37.1	35.2 (15)	29.1 (21)	43.6 (18)	42.2 (15)	33.2 (20)	47.2 (19)	33.1 (17)	32.4 (16)	43.5 (19)	41.3 (20)	35.5 (16)
TLD [24]	18.1	35.9	29.0 (19)	35.1 (19)	44.4 (17)	32.5 (19)	38.5 (17)	51.6 (16)	29.7 (20)	33.8 (15)	38.7 (20)	50.2 (17)	30.5 (20)
MIL [1]	18.5	35.5	32.2 (16)	37.2 (18)	38.3 (19)	36.6 (18)	34.6 (18)	45.5 (20)	31.5 (19)	25.6 (19)	49.0 (17)	40.4 (21)	33.6 (19)
SemiB [40]	20.6	28.3	22.5 (21)	33.0 (20)	32.7 (21)	24.0 (21)	31.6 (21)	38.2 (21)	24.4 (21)	25.1 (20)	32.7 (21)	41.9 (19)	23.2 (21)
OF [39]	22.0	18.6	17.9 (22)	11.4 (22)	23.4 (22)	20.1 (22)	17.5 (22)	18.1 (22)	18.8 (22)	15.9 (22)	22.3 (22)	23.4 (22)	16.8 (22)

Fig. 13 Difficult segmentation and tracking scenarios sequence *face_move1* (**a**) and sequence *wuguiTwo_no* (**b**). *Red line* DS-KCF, *blue line* DS-KCF$_{\text{shape}}$

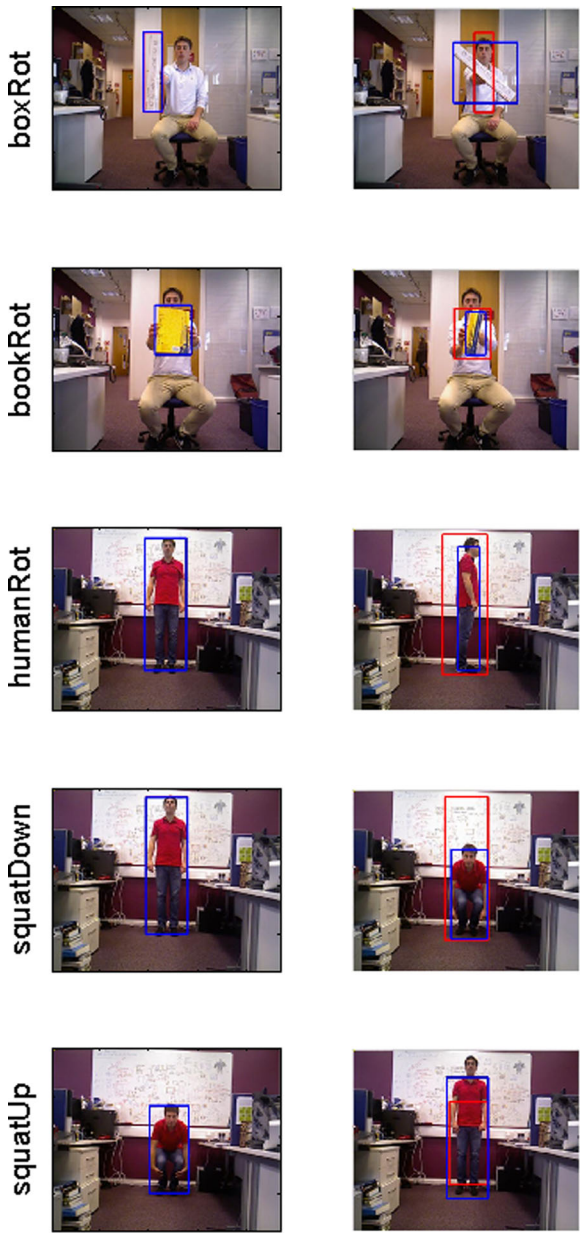

Fig. 14 RotTrack shape change sequences and tracking output *before and after* change of shape. *Red bounding box* DS-KCF, *blue bounding box* DS-KCF w. shape

Table 6 Performance of the DS-KCF with and without the shape handling module on the RotTrack sequences

Sequence	Method	AUC	P20
boxRot	DS-KCF	**74.4**	**96.3**
	DS-KCF no shape	41.6	83.4
bookRot	DS-KCF	**72.3**	96.9
	DS-KCF no shape	54.6	**100**
humanRot	DS-KCF	**83.8**	98.6
	DS-KCF no shape	70.6	**100**
squatDown	DS-KCF	**83.1**	**96.2**
	DS-KCF no shape	73	46.7
squatUp	DS-KCF	**79.1**	**44.6**
	DS-KCF no shape	68.9	36.3
Average	DS-KCF	**78.6**	**86.5**
	DS-KCF no shape	61.7	73.3

All measures are in percentages

planar rotation of a rigid object, non-planar rotation of a rigid object and a person, and deformable object shrinking or expanding (via a person performing simple squatting). The left column of Fig. 14 shows the initial tracked objects in a blue bounding box defining the target object. The second column shows the tracker output during the shape change where the red bounding box is DS-KCF with the shape module switched off, and the blue bounding box is the full DS-KCF. Clearly, the shape handling module provides a more accurate result and a tighter fit on the target object in all these scenarios. Table 6 provides the AUC and P20 results for each of the five sequences. On average, the aspect ratio shape analysis helps to achieve an increase in accuracy of 16.9 and 13.2% for AUC and P20, respectively.

5.5 Discussions

Our results demonstrate that the proposed approach establishes the best trade-off between accuracy and processing speed in comparison to other state-of-the-art RGB-D trackers. A small reduction in tracker accuracy ($\approx$4% on the AUC) is balanced by improvements in processing speed of 180 times, giving real-time responses.

The results illustrate that DS-KCF's accuracy can be improved to manage object shape changes, without dramatically compromising its processing rate. We exploit fast depth-based segmentation to estimate object shape variations without adding additional constraints based on colour features. A trade-off remains between the ability to handle shape variations and occlusion detection. While the proposed extension improves tracking with shape changes, results show increased drift with occlusion.

Finally, it has to be underlined that depth data represent the key to the system's efficiency. It allows us to precisely estimate scale changes at minimal computational cost (see Table 3) and facilitates object segmentation so that its shape may be estimated. However, depth-only segmentation could be a potential drawback of the presented approach as segmentation errors, due to for example noisy depth data or the target interacting with other objects, may cause drifts or enlargement of the tracked object to include neighbouring areas in the environment. As previously discussed, the proposed segmentation and shape estimation module has been formulated so that a very high processing rate may be retained.

6 Conclusions

We presented DS-KCF, a real-time RGB-D tracker, which detects and handles scale changes, manages occlusions, and can deal with shape changes—all facilitated by a fast depth segmentation stage. We analysed a number of different feature combinations and selected a concatenation of HoG and depth features as the best feature set to analyse further. The processing load of the various stages of the proposed method was considered, and the processing rate of the overall method based on its MATLAB and C++ implementations was measured. A full comparison against 19 other object trackers was also presented. DS-KCF performs tracking in real time, processing on average 35.7 fps in MATLAB, and 187 fps in C++ (no-shape handling) when benchmarked on 100 RGB-D sequences in the Princeton dataset [39].

Acknowledgements This work was performed in the SPHERE IRC project funded by the UK Engineering and Physical Sciences Research Council (EPSRC), Grant EP/K031910/1. The study used the Princeton RGB-D data available from http://tracking.cs.princeton.edu. The RotTrack RGB-D data created in-house and the C++ and MATLAB code of the proposed DS-KCF algorithm are available from the SPHERE website at http://www.ircsphere.ac.uk/work-package-2/DS-KCF-JRTIP.

References

1. Avidan, S.: Support vector tracking. IEEE Trans. Pattern Anal. Mach. Intell. **26**(8), 1064–1072 (2004)
2. Awwad, S., Hussein, F., Piccardi, M.: Local depth patterns for tracking in depth videos. In: ACM Multimedia, pp. 1115–1118 (2015)
3. Bolme, D., Beveridge, J., Draper, B., Lui, Y.M.: Visual object tracking using adaptive correlation filters. In: IEEE CVPR, pp. 2544–2550 (2010)
4. Brox, T., Malik, J.: Large displacement optical flow: descriptor matching in variational motion estimation. IEEE Trans. Pattern Anal. Mach. Intell. **33**(3), 500–513 (2011)
5. Camplani, M., Mantecon, T., Salgado, L.: Depth-color fusion strategy for 3-D scene modeling with kinect. IEEE Trans. Cybern. **43**(6), 1560–1571 (2013)
6. Camplani, M., Hannuna, S., Mirmehdi, M., Damen, D., Paiement, A., Tao, L., Burghardt, T.: Real-time RGB-D tracking with depth scaling kernelised correlation filters and occlusion handling. In: BMVC, pp. 145.1–145.11 (2015)
7. Camplani, M., Paiement, A., Mirmehdi, M., Damen, D., Hannuna, S., Tao, L., Burghardt, T.: Multiple human tracking from RGB-D data: a survey. IET Comput. Vis. (2017) (**to appear**)
8. Chen, T., Chen, Y., Chien, S.: Fast image segmentation based on K-means clustering with histograms in HSV colour space. In: IEEE MSP workshop, pp. 322–325 (2008)
9. Chen, Z., Hong, Z., Tao, D.: An experimental survey on correlation filter-based tracking. CoRR abs/1509.05520, http://arxiv.org/abs/1509.05520 (2015)
10. Chrapek, D., Beran, V., Zemcik, P.: Depth-based filtration for tracking boost. ACIVS **9386**, 217–228 (2015)
11. Danelljan, M., Häger, G., Shahbaz Khan, F., Felsberg, M.: Accurate scale estimation for robust visual tracking. In: BMVC, pp. 38.1–38.11 (2014a)
12. Danelljan, M., Khan, F., Felsberg, M., van de Weijer, J.: Adaptive color attributes for real-time visual tracking. In: IEEE CVPR, pp. 1090–1097 (2014b)
13. Du, Q., Cai, Z.q., Liu, H., Yu, Z.L.: A rotation adaptive correlation filter for robust tracking. In: IEEE DSP, pp. 1035–1038 (2015)
14. Galoogahi, H., Sim, T., Lucey, S.: Multi-channel correlation filters. In: IEEE ICCV, pp. 3072–3079 (2013)
15. García, G., Klein, D., Stückler, J., Frintrop, S., Cremers, A.: Adaptive multi-cue 3D tracking of arbitrary objects. In: Pattern Recognition, pp. 357–366 (2012)
16. Gonzalez, R., Woods, R.: Digital Image Processing. Addison-Wesley Longman Publishing Co. Inc, Reading (1992)
17. Gupta, S., Arbelaez, P., Malik, J.: Perceptual organization and recognition of indoor scenes from RGB-D images. In: IEEE CVPR, pp. 564–571 (2013)
18. Gupta, S., Girshick, R., Arbeláez, P., Malik, J.: Learning rich features from RGB-D images for object detection and segmentation. ECCV **8695**, 345–360 (2014)
19. Haag, K.: KCF implementation C++. GitHub repository. https://github.com/klahaag/cf_tracking (2015)
20. Han, J., Shao, L., Xu, D., Shotton, J.: Enhanced computer vision with microsoft kinect sensor: a review. IEEE T-Cybern. **43**(5), 1318–1334 (2013)
21. Hare, S., Saffari, A., Torr, P.: Struck: structured output tracking with kernels. In: IEEE ICCV, pp. 263–270 (2011)
22. 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)
23. Hester, C.F., Casasent, D.: Multivariant technique for multiclass pattern recognition. Appl. Opt. **19**, 1758–1761 (1980)
24. Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-learning-detection. IEEE Trans. Pattern Anal. Mach. Intell. **34**(7), 1409–1422 (2012)

25. Khoshelham, K., Elberink, S.: Accuracy and resolution of kinect depth data for indoor mapping applications. Sensors **12**(2), 1437–1454 (2012)
26. Klein, D., Cremers, A.: Boosting scalable gradient features for adaptive real-time tracking. In: IEEE ICRA, pp. 4411–4416 (2011)
27. Kristan, M., et al.: The Visual Object Tracking VOT2014 challenge results. In: ECCV Visual Object Tracking Challenge Workshop (2014)
28. Kwon, J., Lee, K.: Visual tracking decomposition. In: IEEE CVPR, pp. 1269–1276 (2010)
29. Leal-Taixé, L., Milan, A., Reid, I., Roth, S., Schindler, K.: MOTChallenge 2015: towards a benchmark for multi-target tracking. arXiv:150401942 [cs] (2015)
30. Li, X., Hu, W., Shen, C., Zhang, Z., Dick, A., Hengel, A.: A survey of appearance models in visual object tracking. ACM Trans. Intell. Syst. Technol. **4**(4), 271–288 (2013)
31. Li, Y., Zhu, J.: A scale adaptive Kernel correlation filter tracker with feature integration. ECCV Workshops, vol. 8926, pp. 254–265 (2015)
32. Li, Y., Zhu, J., Hoi, S.C.: Reliable patch trackers: Robust visual tracking by exploiting reliable patches. In: IEEE CVPR, pp. 353–361 (2015)
33. Liu, T., Wang, G., Yang, Q.: Real-time part-based visual tracking via adaptive correlation filters. In: IEEE CVPR, pp. 4902–4912 (2015)
34. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
35. Ma, C., Yang, X., Zhang, C., Yang, M.H.: Long-term correlation tracking. In: IEEE CVPR, pp. 5388–5396 (2015)
36. Meshgi, K., ichi Maeda, S., Oba, S., Skibbe, H., zhe Li, Y., Ishii, S.: An occlusion-aware particle filter tracker to handle complex and persistent occlusions. Comput. Vis. Image Underst. **150**, 81–94 (2016)
37. Scholkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
38. Smeulders, A., Chu, D., Cucchiara, R., Calderara, S., Dehghan, A., Shah, M.: Visual tracking: an experimental survey. IEEE Trans. Pattern Anal. Mach. Intell. **36**(7), 1442–1468 (2014)
39. Song, S., Xiao, J.: Tracking revisited using RGB-D camera: unified benchmark and baselines. In: IEEE ICCV, pp. 233–240 (2013)
40. Stalder, S., Grabner, H., Van Gool, L.: Beyond semi-supervised tracking: tracking should be as simple as detection, but not simpler than recognition. In: IEEE ICCV Workshops, pp. 1409–1416 (2009)
41. Wang, Q., Fang, J., Yuan, Y.: Multi-cue based tracking. Neurocomputing **131**, 227–236 (2014)
42. Weikersdorfer, D., Gossow, D., Beetz, M.: Depth-adaptive superpixels. In: ICPR, pp. 2087–2090 (2012)
43. Wu, Y., Lim, J., Yang, M.: Online object tracking: a benchmark. In: IEEE CVPR, pp. 2411–2418 (2013)
44. Xu, G., Xu, X., Xing, X., Cai, B., Qing, C.: Multi-invariance appearance model for object tracking. In: IEEE DSP, pp. 347–351 (2015)
45. Xu, Y., Wang, J., Li, H., Li, Y., Miao, Z., Zhang, Y.: Patch-based scale calculation for real-time visual tracking. IEEE Signal Process. Lett. **23**(1), 40–44 (2016)
46. Zhang, K., Zhang, L., Yang, M.H.: Real-time compressive tracking. In: ECCV, pp. 864–877 (2012)
47. Zhang, K., Zhang, L., Liu, Q., Zhang, D., Yang, M.H.: Fast visual tracking via dense spatio-temporal context learning. In: ECCV, pp. 127–141 (2014)
48. Zhu, G., Wang, J., Wu, Y., Lu, H.: Collaborative correlation tracking. In: BMVC, pp. 184.1–184.12 (2015)

Sion Hannuna received his Ph.D. in computer vision from the University of Bristol in 2007. His Ph.D. focused on automatically detecting ambulatory quadrupeds in low-quality wildlife videos. He has also developed software to power a cognitive aid system for blind people (CASBliP project). His research is currently focussed on quality of movement assessment and tracking.

Massimo Camplani is a research associate on the SPHERE project at the University of Bristol. His research interests include areas in computer vision. Camplani received Ph.D. in electronic and computer engineering from the Università degli Studi di Cagliari.

Jake Hall is an undergraduate in the Department of Computer Science at the University of Bristol, UK. His interests include computer vision, software development, object-oriented design, and functional programming.

Majid Mirmehdi (MM) is a Professor of Computer Vision in the Department of Computer Science at the University of Bristol and member of the Bristol Robotics Laboratory and the Bristol Vision Institute (BVI). He is the Graduate Dean and Faculty Graduate Education Director in the Faculty of Engineering at Bristol. His research interests include natural scene analysis and health care monitoring using vision and other sensors, and he has more than 200 refereed conference and journal publications in these and other areas. MM is a Fellow of the International Association for Pattern Recognition. He is Editor-in-Chief of IET Computer Vision journal and an Associate Editor of the Pattern Analysis and Applications journal. He is a member of the IET, Senior Member of IEEE, and serves on the Executive Committee of the British Machine Vision Association as well as the IET Vision and Imaging Network.

Dima Damen is a Lecturer in Computer Vision at the Department of Computer Science, University of Bristol, UK. She received the B.Sc. degree ('02) in computer science from Birzeit University, the M.Sc. degree ('03) in distributed multimedia systems, and the Ph.D. ('09) degree in computer vision from the University of Leeds, UK. Her current research interests include event, action, and activity analysis and recognition, shape-based object detection, and egocentric vision. Dima is an Associate Editor of IET Computer Vision (2013), co-chaired BMVC13, an area chair for BMVC (2014–2016). She is a member of IEEE and BMVA.

Tilo Burghardt Tilo Burghardt′s research focus is applied computer vision. He graduated with Distinction in Media Computing at Dresden University of Technology (Germany) and subsequently completed his postgraduate studies at the University of Bristol (UK). He is currently a Lecturer at the Department of Computer Science there. He is a member of the British Machine Vision Association (BMVA) and the German Academic Foundation (Studienstiftung des Deutschen Volkes).

Adeline Paiement is a Research Associate in Computer Vision in the Computer Science Department at the University of Bristol. She received a Ph.D. in computer vision from the same university in 2014 for her work on 3D/4D object reconstruction from sparse and misaligned medical images. She is currently working with the SPHERE Project on movement and activity analysis for health monitoring in a home environment.

Lili Tao is a Research Associate at the Visual Information Laboratory at the University of Bristol. Her research interests include human motion analysis, 3D deformable object reconstruction, and facial expression analysis. Tao received a Ph.D. in computer vision from the University of Central Lancashire.