University of Exeter
Department of Computer Science

Object Tracking in Video with Part-Based Tracking by Feature Sampling

George De Ath

Submitted by George De Ath, to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Computer Science, May, 2019.

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

Signed: .....

Abstract

Visual tracking of arbitrary objects is an active research topic in computer vision, with applications across multiple disciplines including video surveillance, activity analysis, robot vision, and human computer interface. Despite great progress having been made in object tracking in recent years, it still remains a challenge to design trackers that can deal with difficult tracking scenarios, such as camera motion, object motion change, occlusion, illumination changes, and object deformation. A promising way of tackling these types of problems is to use a part-based method; one which models and tracks small regions of the object and estimates the location of the object based on the tracked part's positions. These approaches typically model parts of objects with histograms of various hand-crafted features extracted from the region in which the part is located. However, it is unclear how such relatively homogeneous regions should be represented to form an effective part-based tracker.

In this thesis we present a part-based tracker that includes a model for object parts that is designed to empirically characterise the underlying colour distribution of an image region, representing it by pairs of randomly selected colour features and counts of how many pixels are similar to each feature. This novel feature representation is used to find probable locations for the part in future frames via a Bhattacharyya Distance-based metric, which is modified to prefer higher quality matches. Sets of candidate patch locations are generated by randomly generating non-shearing affine transformations of the part's previous locations and locally optimising the most likely sets of parts to allow for small intra-frame object deformations. We also present a study of model initialisation in online, model-free tracking and evaluate several techniques for selecting the regions of an image, given a target bounding box most likely to contain an object. The strengths and limitations of the combined tracker are evaluated on the VOT2016 and VOT2018 datasets using their evaluation protocol, which also allows an extensive evaluation of parameter robustness. The presented tracker is ranked first among part-based trackers on the VOT2018 dataset and is particularly robust to changes in object and camera motion, as well as object size changes.

Acknowledgements

So long, and thanks for all the fish.

— Dolphins, *The Hitchhiker's Guide to the Galaxy*

Contents

List of Figures

List of Tables

Nomenclature

Images

$I(t)$	Image frame at time t.
$I_i(t)$	Image pixel location.
$\mathbf{x}_i(t)$	Image pixel features located at $I_i(t)$.
Ω_p	Region of image for patch p.

Counts

P	Number of part models (patches).
S	Number of feature samples used in each model.
G	Number of sets of candidate patches generated.
L	Number of sets of candidate patches locally optimised.

Tracker variables

$\mathcal{M}$	All patch models.
$\mathcal{M}_p$	Patch model.
$\mathbf{f}_s$	Feature sample s.
c_s	Number of features in a patch matching feature s.
R	Match radius.
φ	Patch overlap threshold.
$\pi_x,\ \pi_y,\ \pi_r,\ \pi_s$	Probability distributions to sample horizontal (x) and vertical (y) translation, rotation (r) and object scale (s) from.
$\mu_x,\ \mu_y,\ \mu_r,\ \mu_s$	Corresponding location parameters.
$\sigma_x,\ \sigma_y,\ \sigma_r,\ \sigma_s$	Corresponding scale parameters.
$\tilde{w},\ \tilde{h}$	Predicted object width and height.

b	Modified Bhattacharyya Distance parameter.
W	Local optimisation window size.
β_c	Update rate: counts.
β_s	Update rate: samples.

Initialisation techniques

$\mathcal{G}$	Ground-truth mask of object.
$\mathcal{B}$	Ground-truth mask of object restricted to give bounding box.
$\mathcal{P}$	Predicted mask of object.
$\mathcal{S}$	Set of superpixels.
$\mathcal{X}$	(OC-SVM) Pixel feature space.
$\mathcal{F}$	(OC-SVM) Dot product feature space.
γ	(OC-SVM) RBF kernel width.
$\Phi(\cdot)$	(OC-SVM) Implicit function.
ξ_i	(OC-SVM) Slack variable.
ν	(OC-SVM) Regularisation parameter.
S	(SBBM) Number of feature samples.
δ	(SBBM) Proportion of feature samples to include in model.
η	(SBBM) Match threshold.
$Q,\ q$	(SBBM) Pixel and model matching functions.
$F,\ B$	(LBDM) Foreground and background components of image I.
α	(LBDM) Alpha mask.
$\rho^+,\ \rho^-$	(LBDM) Bounding box expansion and contraction.
τ	(LBDM) Proportion of bounding box to be labelled as foreground.
λ	(LBDM) Regularisation parameter.
c	(LBDM) Penalty of deviating from known label.

1. Introduction

Visual object tracking is one of the fundamental challenges in computer vision. It is the task of estimating an object's location over a sequence of images. Despite strong progress having been made in recent years, in part driven by the emergence of popular benchmark datasets (Kristan et al., 2015; Wu, Lim and Yang, 2013) that have led to increases in tracking performance year on year, it still remains a challenge for trackers to deal with difficult tracking scenarios. Particular challenges include camera motion, illumination change, object motion and size change, as well as occlusion and self-occlusion (Kristan et al., 2013).

Object tracking has a wide variety of applications across multiple fields including video surveillance, activity analysis, robot vision, and human-computer interfaces (Yilmaz, Javed and Shah, 2006). The approaches developed to address these applications depend greatly on the application itself and what *a priori* assumptions can be made. In specific tracking scenarios, such as tracking pedestrians or faces in a crowd, prior knowledge can be exploited to create both visual and motion models of the object. This type of tracking scenario is able to use offline learning in order to construct a model of the tracked object prior to tracking it.

In contrast to offline tracking, in the online tracking setting the classes of the tracked objects are not known, meaning that neither visual nor motion models can be constructed. This is generally referred to as the *model-free* characteristic (Kristan et al., 2015). In this more general setting, trackers are initialised in a region of the first frame of a video sequence, normally defined by a bounding box, that contains the object. This region is created by in one of two ways: by manual initialisation in the form of a user selecting the region to be tracked; or by a specific object detector. In subsequent frames, tracking proceeds based on the initial frame without the

acquisition of additional information about the object. This form of tracking allows the expensive, in terms of human or computational time, task of object identification to only be performed once per video, and automates the tracking of the object.

One of the most commonly overlooked areas of model-free tracking is the problem of determining which pixels within the region to be tracked, provided by the manual initialisation, actually correspond to the object (De Ath and Everson, 2018). For trackers that represent the whole region with one model, for example a correlation filter (Henriques et al., 2015), this is less of an issue as the majority of the region given to the tracker (generally over 70% according to the authors of the VOT2016 benchmark, Vojíř and Matas, 2017) contains pixels belonging to the object. This means that the region-based trackers will have a relatively small proportion of spurious pixels in their model(s). However, for part-based trackers, which represent the object as a set of parts, placement of these parts is more important. If an object part is initialised in an area of the given region that belongs to the background, rather than the object, this part may match a stationary region in the image, rather than the moving object. This can lead to some parts trying to follow the object, while others remain stationary, "tracking" part of the background, resulting in a potential reduction in tracking performance.

The task of tracking objects is trivial to humans, we can easily keep track of multiple objects in our vision. For example, when driving cars on motorways we keep within our lane, sufficient distance from the car in front, and also keep an eye on other cars in the adjacent lanes to avoid accidents. In contrast to this, finding accurate algorithms for the model-free object tracking problem is far more difficult due to the challenging nature of the scenarios encountered. These challenges include tracking objects that are unpredictable and non-linear in their movement, and that change appearance rapidly due to out of plane rotation, self-occlusion and motion blur. The background of the tracked objects can also change rapidly due to perspective changes as the camera moves, depending how far away the foreground object is from its background. Rapid illumination changes frequently occur in unconstrained tracking as objects move in and out of shade and as artificial lighting is turned on an off.

Tracked objects can also quickly change size by either them moving away or towards the camera or the camera itself zooming in or out on the object.

Object tracking methods perform well in simplified tracking scenarios, where there is a fixed camera filming a scene with a static background and constant lighting (Kristan et al., 2013). These types of scenes can also be segmented, separating the moving foreground from the constant background, by using background subtraction algorithms (Xu et al., 2016). These algorithms model the background of an image (or sequence of images) and compare the pixels in future images to their background model. If these pixels are sufficiently similar to their background models they are marked as belonging to the background of the image, if not they marked as belonging to the foreground. This segments an image into foreground and background, where, in the object tracking case, the foreground would likely contain the object.

ViBe (Barnich and Van Droogenbroeck, 2011), a non-parametric pixel-based algorithm, is one of the simplest and most effective background subtraction techniques and performs well on state-of-the-art benchmarks (Xu et al., 2016). Unlike several of its counterparts, it is initialised only on the first frame of video in a sequence, creating a simple background model for each pixel in the image. It does this by extracting image features, typically RGB values, from a pixel and its neighbours and uses them to represent the underlying feature distribution of the background. In subsequent frames, each pixel in the image is compared to its background model, and labelled as belonging to the background if it matches (i.e. is sufficiently close to) a number of the feature samples included in its background model. This allows for objects to be segmented from their background when the scene is sufficiently static. However, if the background is less static, such as when the background or camera is in motion or there are rapid illumination changes, these segmentation methods will fail as the majority of the image does not match a background model and may therefore be labelled as foreground.

1.1 Approach

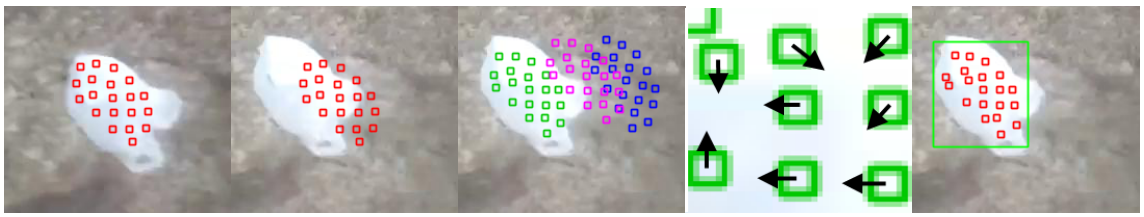
In this thesis, we extend the basic principle of the ViBe background subtraction algorithm, using samples of features to model pixels, and apply it to model-free object tracking. We do this by reversing the background subtraction problem and, instead of modelling an image's background, we model its foreground (i.e. the object). Instead of creating a model for each pixel of the object, we replace the pixel-based representation with a part-based one, using samples of features taken from image patches to characterise the feature distribution of the patches. We develop an object localisation scheme to find the modelled object's parts in subsequent frames, along with an update scheme to update the models once their locations have been predicted.

Additionally, we investigate a problem in model-free object tracking we call the *Initialisation Problem*. This is the task of determining which pixels of the region given in the model-free tracking problem contain the object, and which belong to its background. This is treated as a missing labels problem, because we can be sure that pixels sufficiently far away from the region do not belong to the object and can be therefore labelled as belonging to the object's background. Thus we can define the problem as learning the label (object or background) of pixels inside and close to the region said to contain the object.

1.2 Contributions

We propose (i) a novel object part model that represents an image patch by a set of features and associated counts extracted from them, (ii) a method for selecting initial patch locations, and (iii) an object localisation scheme to address the initialisation problem. We also propose three object segmentation schemes to identify pixels, belonging to an object, within and around a bounding box, based solely on one image and a bounding box denoting the object's location.

We first present a part-based, single-target, tracking algorithm, which outperforms all other part-based methods on the VOT2018 benchmark (Kristan et al., 2019). We model an object with a set of patches distributed over it, and track these



(a) (b) (c) (d) (e)

Figure 1.1: An overview of the proposed tracking algorithm. Given the patch locations in the previous frame (a), and based on their location in the current frame (b), we generate sets of candidate patch locations (c). These are evaluated and the best sets of patch locations are locally optimised (d), and, once re-evaluated, the best set of these are use as the predicted location of the object (e).

patches in subsequent frames of video. Each patch is represented by a set of pairs of features and their associated match counts. A match count denotes how many pixels there are within the modelled patch that match (i.e. have features that are sufficiently close to) the corresponding feature in the model. This representation can be thought as an empirical histogram, consisting of spherical bins in feature space that characterise the underlying feature distribution of the image patch.

Given a region denoting where the object lies, we segment this area into superpixels and initialise patch models in the centre of each of them. The patches (Figure 1.1a) are tracked in subsequent frames (Figure 1.1b) via a novel, two-part, object localisation scheme. This first generates candidate sets of patch locations based on non-shearing affine transformations of the patches' previous locations (Figure 1.1c). The patches in the best sets of these are then locally optimised (Figure 1.1d) within a small window centred on their transformed locations. The object's new location is then predicted as being that of the set of candidate patches with the highest match quality. We perform extensive evaluation of the tracker's components and their parameters on the VOT2016 benchmark (Kristan et al., 2016a), as well was comparing it to other tracking methods on both the VOT2018 (Kristan et al., 2019) and OTB (Wu, Lim and Yang, 2015) benchmarks.

The Initialisation Problem represents the task of determining which pixels in an image, given an approximate location of an object, belong to the object and which do not. We adapt three machine learning techniques to the problem; the first two, One-Class SVM (Schölkopf et al., 2001) and a novel Sampled-Based Background

Model, treat the problem as a one-class classification problem. The third, is an adapted solution to the image matting problem (Zheng and Kambhamettu, 2009). This models each pixel in an image as a proportion of both of foreground and background colour, and aims to learn these proportions. We evaluate these methods on the VOT2016 benchmark, because human-annotated pixel-wise segmentations exist for the dataset, and compare them to predicting the entire bounding box as belonging to the object (as this is effectively what part-based trackers currently do). We show that the alpha matting-based technique is capable of differentiating between an object and its background, much better than assuming the entire bounding box is background, and use it as part of our aforementioned tracker to initialise the tracker's part location.

For clarity, we summarise the contributions below:

- A part-based tracker consisting of a (i) sample-based object part model, (ii) object part initialisation and localisation schemes, as well as a (iii) method for selecting initial patch locations (Chapters 3 and 4).
- Three proposed solutions to the initialisation problem (Chapter 5).

Note that Chapter 5 is based on the work first published in:

- G. De Ath and R. Everson (2018). 'Visual Object Tracking: The Initialisation Problem'. In: *Conference on Computer and Robot Vision (CRV)*, pp. 142–149.

1.3 Overview

The thesis is organised as follows:

Chapter 2 starts by providing a general overview of single-target tracking and then focuses on part-based visual object tracking. The chapter reviews part-based trackers in terms of their object part representation, including features used and matching methodologies, and how they select the initial locations of the constituent parts comprising the object. It reviews the model adaptation schemes used by trackers and how they locate their part's locations in subsequent frames of video. The chapter finishes with a review of the popular tracking benchmarks used in the thesis and the performance measures they use.

Chapter 3 presents a novel, single-target, part-based, tracking algorithm. The chapter describes its sample-based object part representation and how it compares candidate patch locations to its model, as well carrying out an extensive parameter and component evaluation using the VOT2016 benchmark. The chapter also describes the tracker's object localisation scheme (i.e. how it searches for the object in subsequent frames), investigates object motion in the VOT2016 dataset, and performs a parameter and component evaluation for the object localisation component.

Chapter 4 continues the description and evaluation of the tracker's components, and starts by describing the method of selecting the best locations to initialise the tracker's patches in the first frame of video. It presents the patch model initialisation and model update schemes, and shows their parameter and component evaluations. Lastly, the chapter investigates patch drift, the problem of poorly matching patches trying to track parts of the image that do not belong to the object.

Chapter 5 introduces and formulates the Initialisation Problem, and then presents three techniques for addressing it. These are empirically evaluated using the VOT2016 benchmark and compared to the typical part-based tracker's assumption of predicting the entire bounding box as belonging to the object.

Chapter 6 evaluates the tracker using the optimal parameters for each component selected in the previous three chapters. An ablation study carried out on the VOT2018 benchmark and the tracker is both quantitatively and qualitatively evaluated on the VOT2018 and OTB benchmarks.

Chapter 7 concludes the thesis and summarises its contributions. It also provides a discussion on potential directions of future work.

2. Background

Object tracking is an important component of many computer vision tasks, such as video surveillance, activity analysis, robot vision, and human-computer interfaces (Yilmaz, Javed and Shah, 2006). Although great strides have been made in tracking performance in recent years (Kristan et al., 2019), visual object tracking still remains a challenging problem because of the many difficult real-world tracking scenarios that can be encountered. These include camera motion, illumination change, object motion and size change, as well as occlusion and self-occlusion (Kristan et al., 2013). This background chapter gives a brief overview of single-target tracking and the key components of single-target tracking algorithms in Section 2.1. A review of the tracking datasets and evaluation methodologies is also provided in Section 2.2.

Given the sheer scale and popularity of visual tracking approaches over the last two decades, a comprehensive literature study is outside the scope of the thesis and we, instead, focus on establishing the necessary background needed for this thesis. However, there is a large number of literature survey papers published, such as the works of Kristan et al. (2016b), Chen, Hong and Tao (2015), Ojha and Sakhare (2015), Smeulders et al. (2014), Li et al. (2013), Zhang et al. (2013), Salti, Cavallaro and Stefano (2012), Yang et al. (2011), Moeslund, Hilton and Krüger (2006), Yilmaz, Javed and Shah (2006) and Porikli (2006), and we direct the reader towards these surveys for a more expansive overview.

2.1 Single Object Tracking

Tracking algorithms are usually classified as one of two types of methods, depending on the amount of prior information available about the object to be tracked. If the

specific object to be tracked, or even if just its class (e.g. person or car), is known *a priori*, then this additional information can be incorporated into a model of the object. These models can include information about its visual appearance, motion, or shape. Using the additional information as constraints to the problem leads to very specialised tracking methods suitable for only one type of problem. Examples of this include vehicle tracking (Barcellos et al., 2015; Jazayeri et al., 2011; Mei and Ling, 2011), pedestrian tracking (Fragkiadaki et al., 2015; Nam, Dollár and Han, 2014; Ouyang and Wang, 2013; Dalal and Triggs, 2005), and eye tracking (Majaranta and Bulling, 2014; Corcoran et al., 2012; Hansen and Ji, 2010).

Model-free methods, also known as class-free or category-free methods, do not have any prior knowledge available and therefore do not apply any pre-learned models. However, this means that they are far more generalisable because they do not rely on prior information, and instead can be immediately used to track arbitrary objects without the need for a prior training phase. Model-free trackers have, therefore, become popular in recent years, having spawned several popular benchmarks (Kristan et al., 2015; Wu, Lim and Yang, 2013), as they can be used across many different application domains. These benchmarks describe the model-free tracking problem as giving a tracking algorithm the task of tracking an object based on the first frame of a video sequence and a bounding box indicating where the object resides within the frame. These methods are almost always *online* trackers, also known as *causal* trackers, that only use the current and previous frames of video to predict the location of the tracked object. This is a second important characteristic because many systems that have an object tracking component require real-time operation, e.g. human computer interfaces, robotics, and autonomous vehicles. Given the generalisability of the model-free, causal, tracking methods, and the availability of well-respected benchmarks, such as the VOT Challenges (Kristan et al., 2015), we focus on this tracking scenario.

Tracking algorithms that are designed for the model-free tracking scenario consist of several key components. They have a model that can be adapted to any object and a method of comparing this representation to a candidate location of the object

in an image. In order to update the object model with newly observed information about the object in subsequent frames of video, they also have an adaptation, or update, scheme. Part-based algorithms also have additional considerations to take into account, such as how its modelled parts are geometrically related to one another, and the problem of part drift (i.e. when a part is no longer useful for predicting the object's location). The selection of where to place the object's parts in the first frame of video, with respect to a given bounding box, is challenging as one cannot know *a priori* which pixels in the bounding box belong to the object. We, therefore, review work related to each of these aforementioned tracker components.

2.1.1 Object Representation

The way an object is represented, i.e. its model, is arguably the most important component in a tracking algorithm. Objects can either be represented in a holistic manner (Comaniciu, Ramesh and Meer, 2000; Sun et al., 2018; Hare et al., 2016; Henriques et al., 2015; Nam and Han, 2016; Danelljan et al., 2015b; Danelljan et al., 2016; Danelljan et al., 2014a), with a part-based representation, (Battistone, Petrosino and Santopietro, 2017; Akin et al., 2016; Yi et al., 2015; Du et al., 2016; Du et al., 2017; Lukežič, Zajc and Kristan, 2018; Cai et al., 2014; Zhu et al., 2015; Artner, Ion and Kropatsch, 2011; Kwon and Lee, 2009; Maresca and Petrosino, 2015; Maresca and Petrosino, 2013) or a combination of the two (Čehovin, Kristan and Leonardis, 2013; Xiao, Stolkin and Leonardis, 2015; Čehovin, Leonardis and Kristan, 2016a). Holistic methods represent the object using one global model that characterises the entire region that the object resides in, typically its bounding box. Part-based methods use smaller, localised, models to represent sub-regions of the object, known as parts, which can then be used to estimate the object's overall location. Hybrid methods use both global and local appearance models in an attempt to leverage the main advantages of both approaches.

In recent tracking benchmarks (Wu, Lim and Yang, 2013; Kristan et al., 2016a; Kristan et al., 2019) holistic models have been proven to perform well, as they are able to track successfully under the effects of blur and illumination changes (Čehovin,

Kristan and Leonardis, 2013). However, due to their singular representation, when objects deform in a way that changes their appearance they can suffer from poorer tracking performance, which in turn can lead to tracking failure. If an object deforms rapidly over several frames, the holistic models may drift partially off the target as they fail to exactly predict where the object is located. This can result in them updating their model of the object with background information and consequently increasing the likelihood of trying to match the background instead of the object in subsequent frames. Conversely, part-based methods tend to perform better in the presence of deformation because their parts can move independently (to differing degrees depending on the specific method), and can therefore follow parts of objects that move in different directions to one another. However, the part models may become detached and start to drift, tracking something other than the object. This can lead to an over-estimation of the object's bounding box as well as potentially leading to tracking failure as more patches drift over time.

The models that both holistic and part-based methods use to represent the object (or its parts) are very similar, as part-based methods tend to use object part models that are smaller versions of the holistic models. Trackers generally represent an object using either keypoints, image patches (regions), correlation filters, or Convolutional Neural Networks (CNNs). For example, Henriques et al. (2015) use a correlation filter as their holistic model, and Liu, Wang and Yang (2015) use the same correlation filter for each of their object part models. In the following section we briefly review each of these object models and how they are compared to a potential object location. Since we are proposing a part-based tracking algorithm, we also review each method with respect to its use in part-based tracking. This is followed by an overview of the types features used in each of these models.

Keypoint Methods

These methods represent an object using a set of feature points, such as ORB (Rublee et al., 2011), SIFT (Lowe, 1999), or SURF (Bay et al., 2008), and perform correspondence matching with them, i.e. they try to find the keypoints in subsequent

frame. The features used to describe these points are chosen such that they have desirable properties for matching, e.g. invariance to image scale and rotation for SIFT features (Lowe, 1999). The features are extracted from locations, from within the region of interest, that are selected in order to produce highly distinctive features that very few other locations in the image match well to. Once extracted in one frame of video, trackers look for correspondences in a subsequent frame of video, usually getting multiple matches for each keypoint. Therefore, some form of structure, affine motion for example, has to be imposed. This is done in order to reject matching keypoints that have falsely matched a region in the image that does not belong to the object, and can be carried out using an outlier detection algorithm such as RANSAC (Fischler and Bolles, 1981),

Holistic trackers infrequently use keypoints, however some use them as a long-term component in their method to aid in re-detection of the object if it is occluded (Hong et al., 2015b), or to estimate scale change (Montero, Lang and Laganière, 2015). Keypoint-based methods are traditionally considered to be a part-based method (Wu, Lim and Yang, 2013; Kristan et al., 2013; Kristan et al., 2015; Kristan et al., 2019) because they track a set of parts, i.e. keypoints, and use them to estimate an object's location. MATRIOKSA (Maresca and Petrosino, 2013), for example, uses multiple types of keypoint features in an attempt to increase robustness and uses a Generalised Hough Transform to model the keypoint correspondences between frames. CMT (Nebehay and Pflugfelder, 2015) determines which keypoints are valid correspondences by using them to predict the object's centre of mass. An outlier detection scheme is used in order to identify and discard those keypoints whose centre of mass prediction disagrees with the majority vote.

Convolutional Neural Networks

In recent years, Convolutional Neural Networks (CNNs) have greatly advanced the areas object recognition and detection (Razavian et al., 2014; Ren et al., 2015; Redmon et al., 2016; Redmon and Farhadi, 2017). As a consequence of this, various approaches have been developed based on CNNs and other deep learning methods

for single-target object tracking. Several methods (Nam and Han, 2016; Song et al., 2017; Hong et al., 2015a; Wang et al., 2016) make use of the discriminative deep features found in the convolution's layers of a CNN. These typically make use of pre-trained CNNs, such as AlexNet (Krizhevsky, Sutskever and Hinton, 2012) or VGG (Simonyan and Zisserman, 2015). However, others, such as MDNet (Nam and Han, 2016) use multi-domain learning to train a CNN for the task of object tracking by having the network learn how to predict an object's bounding box from an input image for a given image sequence. It repeatedly relearns this process for each training image sequence, removing the final layer(s) of the network (known as domain-specific layers) after each sequence. This allows the earlier layers of the network to learn about the generic tracking problem, as the later, removed, layers are thought to only contain information related to the specific tracked object.

Recurrent Neural Networks (RNNs) and Siamese networks have also recently gained popularity within the object tracking community (Bertinetto et al., 2016b; Cui et al., 2016; Fan and Ling, 2017; Kosiorek, Bewley and Posner, 2017). The majority of Siamese CNNs are based on the methodology of Bertinetto et al. (2016b), who train a Siamese network to locate an image of an object from within a larger image also containing the object. All three types of CNN-based methods take in the image frame containing the object to be tracked as input, and typically return the coordinates of a bounding box they predict contains the image. The may also make use of a bounding box regressor (e.g. Girshick et al., 2014) in order to find tighter bounding boxes that better enclose the target (Nam and Han, 2016).

As far as we are aware, no existing part-based trackers use CNNs as their object part models, and CNN-based trackers usually ignore the spatial layout of the target object. However, LSART (Sun et al., 2018) uses a spatially-regularised convolutional kernels in its CNN, each of which focus on a specific region of the target. While this enables it to weight the importance of different regions of the target object, we do not consider this to be a true part-based technique as the parts are fixed and cannot move with respect to each other.

Image Patches

Characterising an object or object parts as a rectangular image patch is one of the most commonly used representations in tracking (Li et al., 2013). Patches are usually represented by either templates or histograms. Templates are essentially just a cropped version of the image, with each location in the template containing the intensity value of the corresponding pixel. A candidate object's location is then compared to the template by either summing the absolute differences between the template and an identically sized region centred on the potential object location (an ℓ_1 measure), or by measuring their cross-correlation. Cross-correlation is an ℓ_2 measure of similarity between vectors (or flattened templates) as a function of the displacement one vector compared to the other.

Histograms capture the feature distribution of an image region, where each bin in the histogram corresponds to a region of the feature space. These have been widely used in object tracking (Adam, Rivlin and Shimshoni, 2006; Čehovin, Kristan and Leonardis, 2013; Lukežič, Zajc and Kristan, 2018; Kwon and Lee, 2009; He et al., 2013; Yi et al., 2015; Artner, Ion and Kropatsch, 2011), although the specific features used to represent the object varies. These are discussed in more detail later on in this section. The histograms of a candidate object (or patch) location and the modelled object are compared to determine their similarity. This is carried out in various ways depending on the specific tracker, however one of the most popular ways of doing this is to use the Bhattacharyya distance, which is based on the Bhattacharyya coefficient (Bhattacharyya, 1946). It was first popularised by the seminal work of Comaniciu, Ramesh and Meer (2000) and measures the overlap of two probability distributions that are represented by histograms. In this work we use a patch-based representation and compare its match quality using a modified version of the Bhattacharyya distance. Details of both the patch model and modified distance are given in Chapter 3.

Correlation Filters

Correlation filters measure the similarity between an image template and a candidate region in the image. The are designed so that they produce a peak response at the centre of the object of interest and produce a low response elsewhere (Chen, Hong and Tao, 2015). Correlation filters were first popularised by the MOSSE tracker (Bolme et al., 2010) which showed that they can run at high frame rates as well as being robust to illumination changes (compared to other trackers of the time). The circulant matrices extension of the Kernalised Correlation Filter tracker (KCF, Henriques et al., 2015) enabled correlation filters to model all possible translations of an object template via kernel ridge regression in the Fourier domain. Their development has been ongoing in recent years, with further extensions to the representation such as extending the original image intensity-based representation to features that can use an arbitrary number of channels (Galoogahi, Sim and Lucey, 2013; Danelljan et al., 2014a) and also to non-linear kernels (Henriques et al., 2015).

There have been several part-based methods using correlation filters to represent object parts. They typically use the KCF tracker for their object parts (Liu et al., 2018; Liu, Wang and Yang, 2015; Lukežič, Zajc and Kristan, 2018; Akin et al., 2016; Yao et al., 2017), independently training a KCF correlation filter for each object part. However, other methods, such as that of Liu et al. (2016) jointly train filters so they predict object motion that is in the same direction as one another.

Correlation filters have been used to great success in object tracking (Kristan et al., 2019), but do suffer performance issues when their the template size is small (Lukežič, Zajc and Kristan, 2018). This is because the search range of a correlation filter is limited by its size, due to only being able to measure correlation up to half its width or height away from where it is centred in the image.

Feature Representations

Correlation Filter and image region/patch-based methods use similar types of histogram-based feature representations. Recently, features extracted from the convolutional activation layers in CNNs that have been pre-trained on large-scale

image classification tasks have been used successfully in correlation filter-based tracking, such as the imagenet-vgg-2048 network (Chatfield et al., 2014) in the work of Danelljan et al. (2015a). Correlation filters that use these have been some of the top performing on the recent VOT2018 benchmark (Kristan et al., 2019). The two other main types of features that are used throughout the tracking community are colour and edge/texture-based features.

The colour of an object is widely used to describe object appearance, and has been used to great effect to track objects (Li et al., 2013; Collins and Leordeanu, 2005; Nummiaro, Koller-Meier and Gool, 2003). One of the most notable early works was that of the mean-shift tracker (Comaniciu, Ramesh and Meer, 2000) which first introduced a metric based on the Bhattacharyya coefficient (Bhattacharyya, 1946) for comparing two histograms of colour. The work of Danelljan et al. (2014a) first introduced the Color Names (van de Weijer et al., 2009) features to visual object tracking and compared them to many different colour-based features. Color Names features are an automatically learned mapping from RGB colour space to a probabilistic 11 dimensional colour representation consisting of bins corresponding to linguistic colour labels, such as red, green, and blue. These are thought to be somewhat photometrically invariant, as shown by Danelljan et al. (2014a), as the model maps colours, regardless of their illumination, to the same label.

Texture and edge-based features are also important in visual tracking (Yuan, Gao and Xu, 2011; Takala and Pietikainen, 2007; Dalal and Triggs, 2005; Ma, Chen and Chen, 2011; Bertinetto et al., 2016a; Danelljan et al., 2014b; Henriques et al., 2015; Lukežič et al., 2018). They are derived from image gradients (Ojala, Pietikäinen and Mäenpää, 2002) and are often used as an alternative to, or in addition to, colour features. Texture can provide additional information as it describes how the image intensities or colours are spatially arranged. Given the multitude of colour and texture-based features, we refer the reader to the book on colour appearance models by Fairchild (2005), and a recent review of texture-based features by Simon and Vijayasundaram (2018). We now briefly review the three colour and two texture features used in this thesis.

RGB This is an additive colour model that describes the underlying colour distribution of the image region as seen by physical devices. Each pixel in an image is represented by a vector indicating the intensities of the corresponding primary colour (Red, Green, and Blue).

HSV HSV (Smith, 1978) is an alternative to the RGB colour model and is based on a cylindrical coordinate representation of RGB colour space. Each pixel is represented by three channels corresponding to its hue (H), saturation (S), and value (V). Hue, the angle around the central vertical axis, represents the specific colour, whereas saturation and value represent the departure from white and black respectively.

LAB CIELAB, also known as CIE[1] L*a*b*, is a perceptually uniform colour-space, i.e. equal perceived visual changes between two colours correspond to equal distances between the colour-space. It represents colour by lightness (L) and two colour components indicating its position between red and green (A) and its position between blue and yellow (B). Note that for ease of description we refer to CIELAB as LAB.

LBP Local Binary Patterns (LBP) are non-parametric, texture-based, feature descriptors that describe the local structures of images (Ojala, Pietikäinen and Mäenpää, 2002). It labels each pixel in an image by thresholding its immediate neighbourhood and converting the result into a binary number. These numbers are then converted to integers and collected into bins of a histogram, such that pixels with similar labels (i.e. have similar local texture) reside in the same bins. They are thought to be invariant to illumination (Huang et al., 2011) because they only compare pixels on the basis of their relative intensity values.

SIFT The Scale-Invariant Feature Transform (SIFT) is traditionally used for keypoint matching (Lowe, 1999). However, they have been shown to be a powerful feature descriptor when densely extracted from the region of interest (Bosch, Zisserman

[1] *Commission internationale de l'éclairage* (CIE), also known by its English name of the *International Commission on Illumination*, is the Vienna-based international authority on light, illumination, colour, and colour spaces.

and Muñoz, 2006; Bosch, Zisserman and Muñoz, 2007; Vedaldi et al., 2009). SIFT, similar to the HOG feature descriptor (Dalal and Triggs, 2005), is a spatial histogram of 8 oriented gradients. The histograms are extracted across a squared grid of 4×4 windows centred on the point of interest and are spatially weighted, so those farther away from the central point are given less weight. They are then concatenated together to form the SIFT feature vector.

2.1.2 Model Update

In visual object tracking, it is important for the object model to be able to adapt to the object's changing appearance due to factors such scale change, motion blur, lighting changes, and in-plane and out-of-plane rotation of the object (Wang, Hua and Han, 2010). Failure to do so correctly, e.g. if the object's position is poorly estimated, can lead to the object model becoming increasingly dissimilar to the object as more and more erroneous information is included in the object model. This can lead to tracking drift (i.e. the tracker gradually *drifting* away from the object), and eventually tracking failure. This raises the problem of knowing when to update the model, and what parts of it should be updated (Čehovin, Kristan and Leonardis, 2013). If a model is updated too frequently it may include parts of the background due to the object's location being poorly estimated in one frame. Similarly, if the model is updated too conservatively then it will not adapt fast enough to the object's changing appearance. This conflicting objectives problem is known as the template update problem (Matthews, Ishikawa and Baker, 2004), and, despite its name, is applicable to all types of tracking models.

The exact update scheme used in tracking methodologies varies depending on the type of object model used, although they can be roughly separated into those that use template replacement and those that use an autoregressive scheme. Template replacement methods include new observations of the object (and potentially observations which are known to be its background) in their models, replacing older templates with newer ones over time. It is worth noting that trackers using the template replacement strategy do not all use image *templates*, however, the same

methodology, for example, applies to keypoint-based methods which include new keypoints that better describe the object's current appearance (Maresca and Petrosino, 2013). These are, however, usually found in trackers that train supervised learning techniques, such as Support Vector Machines (Hare et al., 2016; Cai et al., 2014), Multiple Instance Learning (Babenko, Yang and Belongie, 2009), Random Forests (Zhang et al., 2017), and decision trees (Xiao, Stolkin and Leonardis, 2015).

The choices of when to include new templates and when to remove older or less useful ones also varies between methods, although a typical scheme tries to maintain a balance between older templates of the object that are known to be accurate and the incorporation of newer ones describing its current appearance. There is also an additional constraint of the number of templates included in the model, as typically this has to be limited in order to track in real-time (Sui, Tang and Zhang, 2015; Hare et al., 2016). Wang, Hua and Han (2010), for example, keep a small proportion of positive and negative examples of the object from the first frame, and replace the others over time. Conversely, Sui, Tang and Zhang (2015) keep the 50 most recent examples and remove older templates in a first-in-first-out buffer.

Autoregressive schemes, also known as exponential smoothers, update their models as a linear combination of the previous model and the most recent estimate of the object:

$$\mathbf{x}_t = (1 - \alpha)\, \mathbf{x}_{t-1} + \alpha\, \tilde{\mathbf{x}}. \tag{2.1}$$

Here $\mathbf{x}_{t-1}$ is some component of the model, typically a histogram of its features, at the previous time-step, and $\tilde{\mathbf{x}}$ is the newly modelled component based on the predicted location of the object. The parameter α is a smoothing term (also known as the the *update rate*) which controls how fast the model updates, i.e. how much much of a memory it has of its previous appearance. Note that $\alpha = 1$ is equivalent to a template replacement scheme that replaces the model component at each time-step. The size of the update rate varies between different trackers, along with whether it is static or dynamic. Nummiaro, Koller-Meier and Gool (2003) and Yi et al. (2015), for example, only update their visual models if it matches the object's estimated location with a high enough certainty, whereas Artner, Ion and Kropatsch (2011) always

update their object part models, but the update rate for each part is proportional to its match quality. Additionally, Cai et al. (2014) combine histograms of the object from the first and current frames of video with a smoothed histogram over the entire image sequence.

Similarly to histogram and template-based approaches, Correlation Filters make use of autoregressive update schemes. They typically update their filter(s) every frame (Bolme et al., 2010; Henriques et al., 2015; Ma et al., 2015; Danelljan et al., 2015b; Lukežič et al., 2018), however Danelljan et al. (2017) argue that updating their filters at each iteration can lead to over-fitting and redundancies in the sample set. They remarked that, ideally, a tracking model should be only updated once significant object change has been detected, but that the problem of what amount of change should be considered significant is extremely difficult. Instead of trying to calculate this with any certainty, they updated their models once for every 6 frames of video processed, and found that it led to an increase in tracking performance compared to updating their model in every frame.

CNN-based approaches update their models depending on the specific type of network and model they are using. Nam and Han (2016), for example, update their CNN with positive and negative training examples when either its best prediction of the object location is of low quality or 10 frames have passed since the last update. The RNN of Cui et al. (2016) is only updated when it predicts the object is not occluded and does so every 5 frames, similarly to the CNN of Song et al. (2017) which updates every 2. However, Siamese Networks have been shown to be able to track objects well without any model update at all, using only one example of the object in the first frame of a sequence (Bertinetto et al., 2016b). In these, the two networks are trained in tandem to predict, given an image with an object to be tracked and an image to locate the object in, where the tracked object lies. We suspect that solving this type of problem naturally leads to the network being somewhat invariant to various types of challenging attributes, such as scale and illumination variation.

2.1.3 Structural Constraints of Part-based Trackers

The way in which part-based methods relate their parts to one another geometrically plays an important role in their tracking performance. The geometric model also greatly influences the ways in which patches can be assessed for poor performance, such as how well they are matching relative to, or if they are drifting away from, their neighbouring patches. Therefore we review common types of geometric models for patch-based trackers and discuss, where applicable, how they handle patch drift.

Several part-based methods enforce no geometric relationships explicitly. Struct-CF (Liu et al., 2016), for example, trains its part models (correlation filters) to predict the same types of motion to give an implicit geometric model, and, the authors say, combat drift. RPAC (Liu, Wang and Yang, 2015) tries to enforce a spatial relationship between its correlation filter-based part model. It does this by having a template with the ideal output from from all parts, modelled on the first frame, that down-weights a part's movement if it it is moving away from the other parts. The authors define patches to be drifting if they are *far away* from the other parts, and places them back into the same spatial configuration in which the parts originally started. However, they fail to clarify what "far away" means in this context. LGCF (Fan and Xiang, 2017) weights its patches' predicted locations in a similar fashion to RPAC, but instead uses a holistic correlation filter to act as a guide to where the patches should be travelling. DPCF (Akin et al., 2016), similarly to LGCF, also uses a larger correlation filter, to guide its parts so as to limit their drifting. Similarly, BHMC (Kwon and Lee, 2009) has no explicit geometric model but removes poorly matching patches, rather than those that drift away, and replaces them with patches located in regions that have similar colour properties to, and that are relatively close to, patches in its model that are matching well.

Other types of part-based trackers enforce geometric constraints on a collection of patches' overall quality by connecting patches with *springs*. These trackers have a geometric likelihood term separate from each patch's visual model that adds additional information into the patch location optimisation process. These springs down-weight a patch's movement if it moves in a different amount or direction to

those it is connected to, as this would either lengthen or shorten the connections between them, thereby acting in a spring-like manner. Yi et al. (2015) propose a spring-based model to deal with non-rigid deformation and do not explicitly deal with drift, but note that spring-like systems implicitly limit drift by controlling how much weight is given to deviation from rigidity (i.e. how stiff the spring is). They use a static weighting for how rigid their model is, which they acknowledge is sub-optimal as, clearly, parts will deform at different rates. DPT (Lukežič, Zajc and Kristan, 2018) takes a different approach and uses a dual spring-system, having its object parts connected to the best matching image locations in addition to being connected to each other. The stiffness of the spring between the part and its best-matched image location is dynamically set based on the quality of the best found image match, allowing for increased deviation from the intra-part spring system if a high quality image match is found. WPCL (Zhu et al., 2015) models the relationship of object parts using a spring-like system, but weights their positions based on how parts were previously configured in relation to one another.

Another way of enforcing spatial consistency is to model the relationships between image regions using graphs, such as the DGT (Cai et al., 2014), SHCT (Du et al., 2016) and GGTv2 (Du et al., 2017) trackers. They model an object with a graph, whose nodes are the centres of superpixels (Ren and Malik, 2003), and try to match this graph in subsequent frames, i.e. matching each superpixel in the previous frame to a superpixel in the current frame. DGT only matches the previous frame's graph to the current frame's, whereas both SHCT and GGTv2 try to enforce geometric relationships over multiple frames. All three methods remove nodes (i.e. superpixels) from the object representation if they are not matched for a certain number of frames, and add new nodes if superpixels are repeatedly unaccounted for within their predicted bounding box of the object.

Trackers can enforce geometric constraints in the way in which they search for the object in an image. The LPT tracker (Yi et al., 2015), which includes a spring system, creates candidate sets of potential part locations by moving them all by a random amount, sampled from a Gaussian distribution and then moving

each part a small random distance. This creates possible object part locations that are almost rigidly affine, with some deviations allowed for local deformation. The tracker combats drifting by only limiting model updates to be carried out when their patches match sufficiently well.

LGT (Čehovin, Kristan and Leonardis, 2013) presents a more advanced part location generation method than LPT. It attempts to learn the optimal affine transformation of the object's part positions in the current frame, followed by learning the optimal local transformation, both carried out using the cross-entropy method. The quality of a set of patches is evaluated by using a visual model and a spatial, elastic graph model that tries to constrain the geometry of the patches to be locally affine, with patches defined as neighbours if they are connected in a Delaunay triangulated mesh. This encodes the idea that neighbouring patches should be moving in a similar manner, but those further away from one another could be moving differently. LGT removes patches that have been matching poorly for several frames, and adds more patches if its estimate of the object's size has increased.

2.1.4 Model Location Initialisation

Knowing which pixels belong to the object, within the given region or bounding box, in the model-free tracking scenario is a challenging problem for part-based methods (Yi et al., 2015). However, holistic trackers, which represent the entire the object by bounding box, find this less of an issue as the majority of their model contains pixels that belong to the object. Some trackers still do make efforts towards acknowledging the relative importance of pixels in the bounding box, such as the mean-shift tracker (Comaniciu, Ramesh and Meer, 2000) that weights the importance of the pixels in the bounding box as being proportional to their distance from its centre. This was mirrored in the seminal work that introduced spatial regularisation of correlation filters (Danelljan et al., 2015b), which also gave more importance to the inner region of the bounding box.

The problem is more pronounced in part-based trackers because object parts can be initialised (i.e. initially placed) in regions within the bounding box that mostly

or completely contain background pixels. This can cause parts to drift away from the object during tracking, as the object, but not its background, moves. Part-based methods typically select the locations of their patches in an *ad hoc* fashion, either distributing them uniformly in some pattern, or trying to select regions that have good properties in relation to their part model.

Those that place their patches uniformly make no assumptions as to which pixels within the patches' locations belong to the object. Some place their component parts uniformly on a grid across the bounding box (Čehovin, Kristan and Leonardis, 2013; Johnander et al., 2017; Yi et al., 2015), while others place their patches so that they touch each other and cover the entire bounding box (Akin et al., 2016; Yao et al., 2017; Lukežič, Zajc and Kristan, 2018; Fan and Xiang, 2017; Adam, Rivlin and Shimshoni, 2006). The tracker of Liu et al. (2016) places its parts such that they overlap by a certain amount and cover the entire bounding box, and the RPAC tracker (Liu, Wang and Yang, 2015) places its parts uniformly spaced out in a cross-like configuration, presumably in an attempt to capture the object in the centre of the bounding box.

Other techniques place their parts based on the type of part-model they have. The ANT tracker (Čehovin, Leonardis and Kristan, 2016a), for example, chooses patch locations that maximise the chance of having good optical flow characteristics because it uses Lucas-Kanade optical flow (Lucas and Kanade, 1981) to estimate the patches' motion between frames. BHMC (Kwon and Lee, 2009) selects areas with good image alignment properties because it warps patches when performing image alignment to estimate patch deformation. Methods that represent object parts as superpixels (Cai et al., 2014; Du et al., 2016; Du et al., 2017) select superpixels that are wholly within the bounding box (i.e. have no pixels overlapping it). The method of Xiao, Stolkin and Leonardis (2015) also segments the bounding box into superpixels, selecting superpixels that overlap the bounding box. They do this because they make the insight that, since superpixeling separates an image into homogeneous regions, any superpixels that contain background are likely to only contain background pixels, meaning that they can quickly prune the poorly matching

superpixels later in tracking.

We address, in Chapter 5, the problem of knowing which pixels, within a given region or bounding box, belong to the object and explore a number of methods to identify object pixels and superpixels.

2.2 Datasets and Evaluation Methodology

Over the last few years the analysis of the performance of tracking has become standardised, with the introduction of datasets, evaluation protocols and competitions aimed at creating sets of challenging tracking videos with real-world tracking scenarios. These include the AVLOV++ (Smeulders et al., 2014), NfS (Galoogahi et al., 2017), NUS-PRO (Li et al., 2016), OTB (Wu, Lim and Yang, 2013; Wu, Lim and Yang, 2015), TColor (Liang, Blasch and Ling, 2015) datasets and the VOT challenges (Kristan et al., 2013; Kristan et al., 2019). We select the two most widely-used methodologies, the *Online Tracking Benchmark* (OTB) and the *Visual Object Tracking challenge* (VOT), to review in this section as they are the most popular short-term, single-target, model-free, causal tracking benchmarks. Since there is considerable overlap between how tracking performance is measured between the various tracking benchmarks, we first give a brief review of tracking performance measures.

2.2.1 Tracking Performance Measures

Tracking performance measures analyse how the tracker's predicted object location compares to the object's actual, annotated ground-truth, location. Although there are many different object tracking performance measures that have gained popularity in recent years (Čehovin, Leonardis and Kristan, 2016b), we focus on those used in the OTB and VOT benchmarks. We start by defining the object's predicted location as being a binary mask $\mathcal{P}$, labelling which pixels within an image it predicts belong to the object, with centroid $\mathbf{c}^{\mathcal{P}}$. Similarly, let $\mathcal{G}$ be the ground-truth binary mask, with centroid $\mathbf{c}^{\mathcal{G}}$, that contains the object's true location determined by human annotators.

Centre Error This measures the difference between the tracker's predicted object centroid and the ground-truth centroid:

$$\Delta\left(\mathcal{P},\mathcal{G}\right)=\left\|\mathbf{c}^{\mathcal{P}}-\mathbf{c}^{\mathcal{G}}\right\|_{2}. \tag{2.2}$$

As it only measures the distance between the two points it is sensitive to the labelling of the object's ground-truth centre and its size, as a small centre error value has different interpretations depending on the object's size. For a small object, e.g. one with sides of length 20 pixels, a 20 pixel centre error would be a complete tracking failure, but for a larger object, e.g. one with sides of 100 pixels, the same centre error size would not indicate failure. Consequentially, tracking failures can have arbitrarily large centre error values.

Region Overlap This measures the overlap between the predicted object's region and the ground-truth annotation and is calculated by the Intersection-over-Union (IOU) measure:

$$\mathrm{IOU}\left(\mathcal{P},\mathcal{G}\right)=\frac{|\mathcal{P}\cap\mathcal{G}|}{|\mathcal{P}\cup\mathcal{G}|}, \tag{2.3}$$

where $\cap$ and $\cup$ represent the intersection and union of two regions respectively and $|\cdot|$ is the region size measured in number of pixels. This is also known as the Jaccard similarity or index and is equivalent to the Dice similarity coefficient (DSC), also known as the F_1 measure, in the sense that $\mathrm{DSC}=2\mathrm{IOU}/(1+IOU)$. The IOU lies in the range $[0,1]$, where a value of 1 represents complete overlap between the predicted and actual location of the object, and 0 indicates that the tracker has drifted off the target object as no overlap exists. The IOU accounts for both position and size of the predicted object's region as well as acting as a failure measure.

Failure Rate This is a count of the number of times a tracker has to be re-initialised after failing to track (i.e. has an $\mathrm{IOU}=0$) an object during one sequence. It mimics the tracker's performance in a real-world scenario where a human operator is supervising the tracker's performance and manually re-initialises it (i.e. corrects its bounding box) when needed. The measure does, however, have several drawbacks.

It does not describe when failures occur, i.e. the distribution of failures across a sequence, or whether a tracker is predicting the entire frame as belonging to the object, in which case its IOU will never equal zero and the tracker will never fail.

2.2.2 Visual Object Tracking Challenge

The VOT challenges (Kristan et al., 2019) provide an evaluation framework containing tracking videos that were selected for their difficulty and the different tracking attributes that they cover. These include occlusion of the object from both itself and its surroundings, illumination change, motion change, object size change and camera motion. They selected the most *difficult* 60 videos from a large pool of tracking datasets (Kristan et al., 2017), given that the aforementioned visual attributes were well represented. Tracking difficulty was estimated by applying three trackers, FoT (Vojíř and Matas, 2014), ASMS (Vojíř, Noskova and Matas, 2013) and KCF (Henriques et al., 2015), to each video and taking an average of their frame average IOU (Equation 2.3) and failure rate over the video to act as a difficulty measure for each video. Full details of their video selection scheme can be found in the VOT2015 challenge paper (Kristan et al., 2015). Each video frame is labelled with the different visual attributes encountered, allowing for detailed analysis to be carried out on tracking performance on a per frame basis.

Trackers are run in a supervised framework 15 times on each video. This allows for statistically significant results to be given and for fairer comparisons to be made between trackers. When the evaluation framework detects a tracking failure (IOU = 0) it re-initialises the tracker 5 frames after the failure to minimise bias being introduced by a particularly difficult segment of a video sequence (Kristan et al., 2016b). Tracking performance is evaluated using three measures, *accuracy*, *robustness*, and *expected average overlap* (EAO).

The accuracy of a tracker is measured by its *average overlap*, i.e. the average IOU (Equation 2.3) of a tracker over a video or dataset. Clearly better performing trackers have higher accuracies. Robustness refers to the average failure rate of the tracker. The appellation *robustness* is somewhat misleading in that better performing

trackers have a lower robustness. The third measure, EAO, first introduced in the VOT2015 challenge (Kristan et al., 2015), was designed to combine both the accuracy and robustness measures into one meaningful statistic. It is an estimator of the average overlap a tracker is expected to achieve over a set of short-term sequences with the same visual properties to those in the dataset.

Note that we use the VOT2016 benchmark (Kristan et al., 2016a) to perform a parameter evaluation of our tracker. VOT2016 contains the same videos as the VOT2015 benchmark but with the addition of more refined ground-truth annotations. In Chapter 6 we perform an ablation study using the optimal parameters found on the VOT2016 benchmark and compare the tracker to other part-based trackers using the VOT2018 benchmark (Kristan et al., 2019). The VOT2018 dataset contains the majority of the same sequences as the VOT2016 dataset, but the 10 least challenging videos have been replaced with more difficult sequences. Given the overlap in videos between the two benchmarks (VOT2016 and VOT2018) there is likely to be at least some over-fitting of the tracker to the earlier benchmark. However, since we select parameters on the basis of EAO value (or being in a range of parameters that produce a high EAO values), we suggest that this is only a minor concern because EAO is an estimator of performance on videos of similar visual attributes and therefore the performance should be independent of the dataset.

2.2.3 Online Tracking Benchmark

The OTB is an older tracking benchmark that focuses on unsupervised experiments (i.e. no re-initialisation). It consists of two datasets, labelled OTB50 and OTB100, that contain 50 and 100 of the most commonly used tracking videos, annotated with per video attribute labels, rather than the per frame annotations present in the VOT challenge. Its standard evaluation protocol is a one-pass evaluation (OPE) in which the tracker is initialised on the first frame of video and is run unsupervised for the rest of the sequence.

Tracking performance is evaluated via the use of success plots and precision plots. A success plot is a measure-threshold plot of the proportion of tracked frames

Benchmark	Videos	Number of Frames			
		Min	Max	Mean	IQR
VOT2018	60	41	1500	356	244
VOT2016	60	41	1500	358	260
OTB100	100	71	3872	590	456

Table 2.1: Video statistics for the three object tracking benchmarks used in this thesis. The number of videos and frame statistics are listed for each dataset, specifically the minimum, maximum, mean, and interquartile range (IQR).

that have an IOU larger than some threshold, ranging from 0 to 1. As the threshold changes from 0 to 1 the success rate changes, resulting in a performance curve. Similarly, precision plots show the proportion of frames that have a centre error of less than a certain number of pixels, ranging from 0 to 50 pixels. Both plots also have associated summary statistics: the area under the success plot is used to represent overall success of the tracker, which also corresponds to the average IOU taken over all frames of video (Čehovin, Kristan and Leonardis, 2014). Precision plots are summarised by the performance of the tracker with a given distance threshold of 20 pixels, i.e. $\Delta(\mathcal{P}, \mathcal{G}) \leq 20$. Babenko, Yang and Belongie (2011) state that a threshold of 20 pixels corresponds to at least a 50% overlap between a tracker's predicted bounding box and the ground-truth and thus can be thought of as a primitive failure rate. We note that this is only the case for smaller objects.

Since all sequences contained in the OTB50 dataset are also contained in the OTB100 dataset, we evaluate our tracker only on the OTB100 dataset.

2.2.4 Datasets Overview

An overview of the three benchmarks used in this thesis is provided in Table 2.1. We note that while the OTB100 benchmark has 100 different objects to track, it actually has 98 unique video sequences. This is because the sequences `Jogging` and `Skating2` both contain two objects to be tracked and are treated separately, known as `Jogging-1` and `Jogging-2`, and `Skating2-1` and `Skating2-2` respectively.

Figure 2.1 shows the object characteristics in each dataset. It is clear from the figures that the the VOT datasets contain harder videos, even though there is some overlap between the OTB and VOT datasets. There is much more scale change

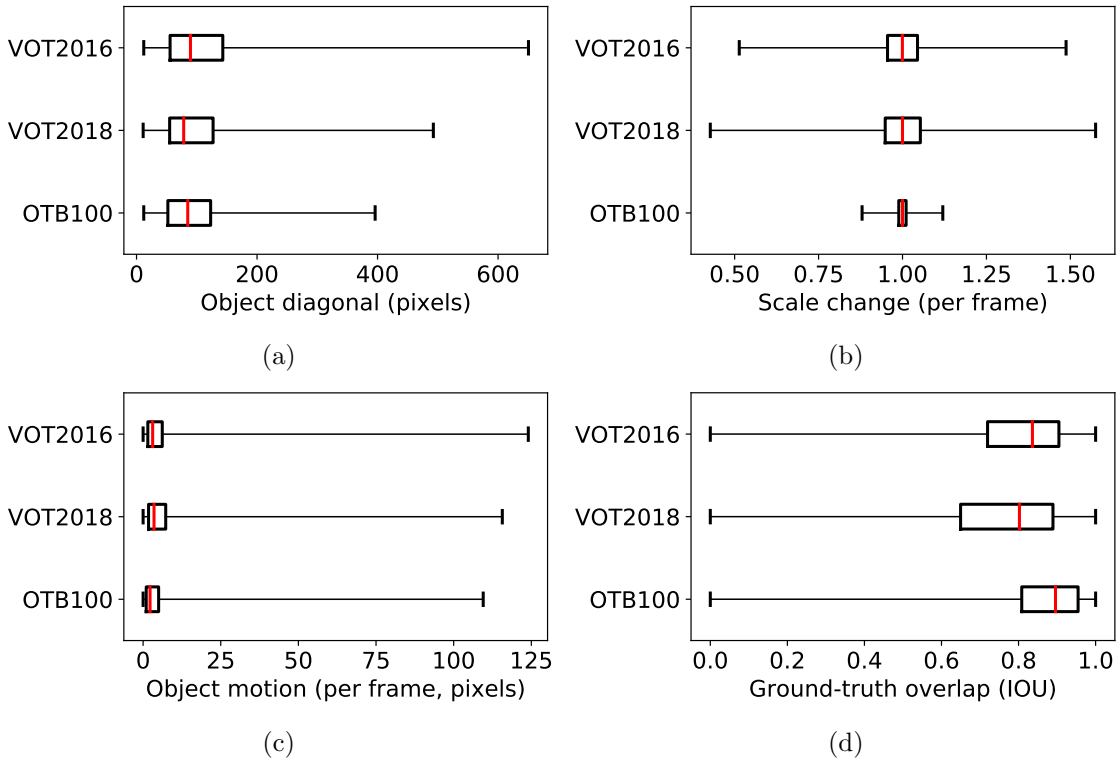


Figure 2.1: Object characteristics for the three benchmarks used in this thesis, showing the distribution of object sizes as described by its bounding box diagonal (a), relative scale change between frames (b), motion between frames (c), and the overlap (IOU) between the ground-truth bounding boxes between consecutive frames (d). Each box plot shows the median (red), the first and third quartiles, as well as the minimum and maximum values on its whiskers. A relative scale change of 1.0 indicates that the object did not change size between frames.

(Figure 2.1b) occurring more frequently, as well as the object's position proving less useful in predicting its future position, as shown by the lower IOU measure between consecutive frames (Figure 2.1d). It is interesting to note, however, that all three datasets have objects of roughly the same size, with an average object diagonal measurement of approximately 100 pixels. This corresponds to square objects of approximately 70×70 pixels or rectangular objects of size 45×90 (assuming a 1:2 aspect ratio). The tails of the box-plots in Figure 2.1d show that the datasets contain challenging sequences because large changes in object location, relative to its size, can cause trackers to fail as they generally assume smaller amounts of (relative) motion.

2.3 Summary

We have briefly surveyed the literature that establishes the necessary background needed for this thesis. We have given an overview of single-target object tracking and have surveyed the main methods for object representation and given a summary of specific feature representations used in the object models. A discussion of how these models are updated and two of the most popular update schemes, template update and autoregressive, were also reviewed, followed by the structural constraints of part-based trackers. We investigated the ways in which part-based trackers place their patches in the first frame of video, given a bounding box indicating where the object is located in the image. Lastly, we gave an overview of the two most popular tracking datasets, VOT and OTB, and discussed their performance measures and evaluation methodology.

In Chapter 3 we first describe the main components of our single-target object tracking algorithm, namely the object representation and localisation scheme, and systematically evaluate their subcomponents and corresponding parameters on the VOT2016 benchmark. Then, in Chapter 4, provide enhancements to the basic tracking model by incorporating patch placement and model initialisation and update schemes, as well as performing an investigation into patch drift. Chapter 5 examines the problem of knowing which pixels, for an image and a given bounding box, belong to an object and which belong to its background and provide several methods for addressing it. An empirical evaluation is then carried out in Chapter 6 in two stages: firstly, an ablation study investigating the performance impact of each component of the tracker is presented. Secondly, our tracker is then compared to other trackers on both the VOT2018 and OTB benchmarks. Finally, in Chapter 7, we summarise the main contributions of this thesis and discuss potential directions for future work.

3. Part-Based Tracking by Sampling

This chapter proposes a new, single-target, part-based tracker for tracking arbitrary objects in challenging video sequences. Overviews of how the tracker operates and how it is evaluated are given in Sections 3.1 and 3.2. The rest of the chapter is split into two main sections, with each of these containing a description of a component of the tracker and associated experimental evaluations: the part-based representation is described in Section 3.3 and the object matching method in Section 3.4.

3.1 Tracker Overview

We first give a high level overview of the part-based object tracking approach in order to give context to the subsequent sections. Figure 3.1 summarises the main stages in the tracking process. As tracking approaches depend on the object model, we first begin by describing how an object is modelled.

An object is modelled by a set of patches, distributed over the object, these are then tracked in subsequent frames of video (Figure 3.2). Each patch is modelled by an empirical distribution of features extracted from the region it represents in the image. For a patch p, this distribution is described by a set $\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_S, C_S) \}$, consisting of pairs of features $\mathbf{f}_s$ and associated counts C_s. The features, usually RGB vectors, are randomly sampled from the image patch and their corresponding counts indicate how many pixels within the patch have features that match the sample. A pixel matches a feature sample if they are within a given radius. In this way, each patch is represented by non-intersecting spheres of feature space, centred

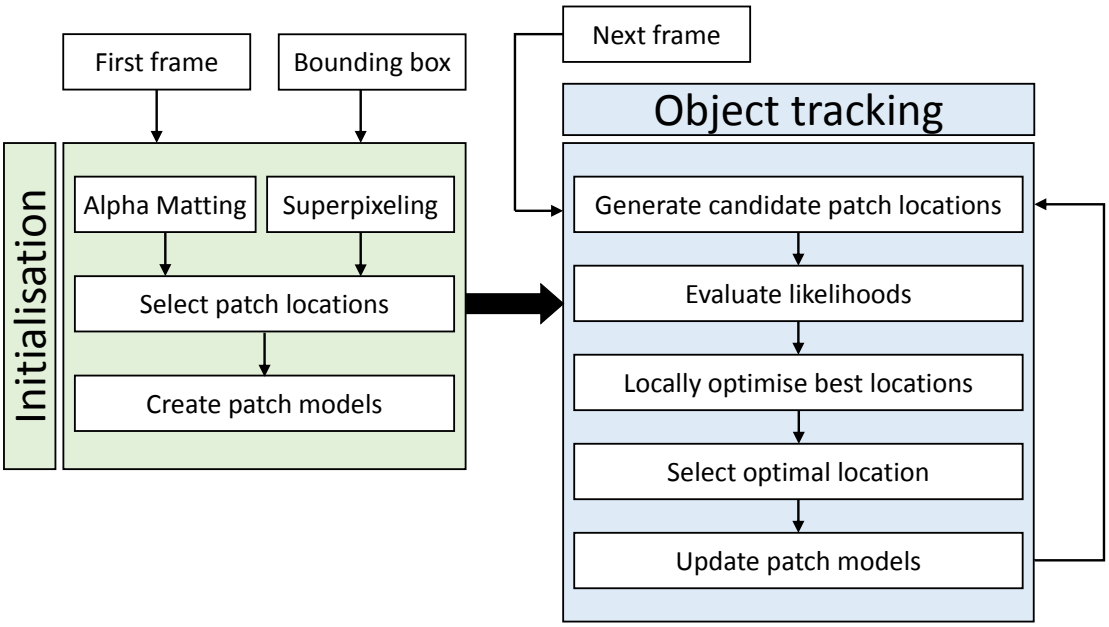


Figure 3.1: Block diagram showing the components of the proposed tracker.

on the model's samples, that can be viewed as an empirical characterisation of the underlying probability distribution of features in the image patch.

Following the short-term, model-free, tracking paradigm (Section 2.1), the object's patch models are initialised using the first frame of a video sequence and the object's bounding box. Since the precise location of the object is not known, the image and bounding box are given to the alpha-matting object segmentation algorithm, described in Chapter 5. This returns a binary mask indicating the regions of the bounding box and its surrounding area that are most likely to contain the object. As described in Section 4.1, the image is then superpixeled, with those superpixels lying outside the masked area being labelled as belonging to the background. Patch locations are then selected as being at the centre of superpixels and their models are initialised (Section 4.2) using the representation described previously.

Once the patch models are initialised, the object is tracked in subsequent frames using a two-part search process (Section 3.4.1). Firstly, candidate sets of patch locations are generated by applying randomly-generated, non-shearing affine transformations to the predicted patch positions in the previous frame of video. Each candidate patch location is compared to its corresponding patch model using a modified version of the Bhattacharyya distance, which gives more weight to higher quality matches. Secondly, the positions of the best matching candidate patch

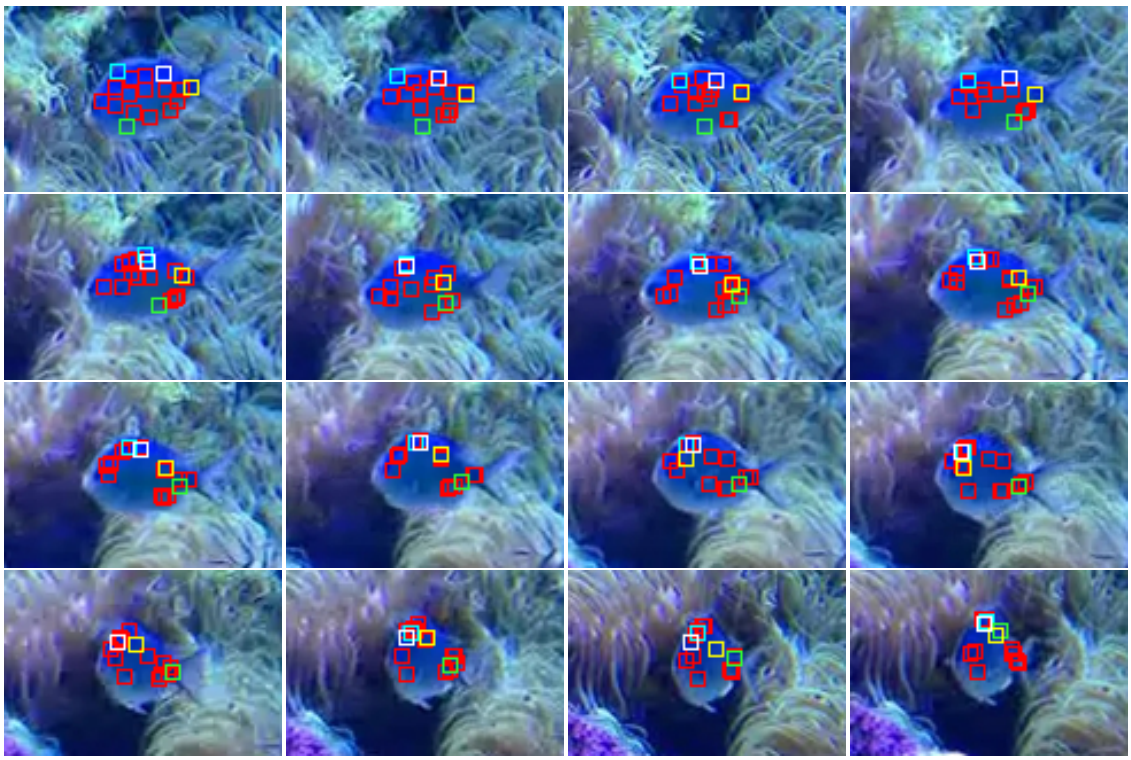


Figure 3.2: Tracking a fish undergoing out-of-plane rotation. These images show a fish in the `fish_1` video, taken from the VOT2016 benchmark (Kristan et al., 2016a), being successfully tracked as it undergoes out-of-plane rotation. To show the movement of the patches (red), images are shown from every fifth frame of the video. Four patches are highlighted in blue, yellow, green, and white, to disambiguate patch motion.

locations are adjusted, within a small window centred on the patch's current location, to maximise the match quality. The set of candidate patch locations that have the highest average match quality are then taken to be the location of the patches, and therefore the object, in that frame.

Lastly, each patch's model is updated (Section 4.3) to to incorporate new information about the object's appearance. The feature and count pairs, $(\mathbf{f}_s, C_s)$ are updated by linearly interpolating between $\mathbf{f}_s$ and the mean feature vector of those pixels that match it, and between C_s and the number of pixels that match $\mathbf{f}_s$. A new patch model is created for those pixels remaining in the patch that did not match any features in $\mathcal{M}_p$, and this is combined with the patch's updated model.

As illustrated in Figure 3.2, this process is iterated for all subsequent frames of the video: candidate sets of locations are generated, evaluated, and locally optimised, with the best matching set of locally optimised locations being used as the new locations for the modelled patches.

3.2 Experimental Overview

In the experimental sections, each of the tracker's constituent parts are evaluated in turn and their parameters are investigated. Their parameters and components are fixed, with one or two varied in order to investigate their influence on tracking performance. The optimal parameters for each tracking component is determined, along with their robustness to deviations from the optimum value. The tracker is evaluated on the VOT2016 challenge (Kristan et al., 2016a), an evaluation protocol that involves repeatedly evaluating the tracker's performance 15 times on each of its 60 video sequences. Each of its video sequences were selected for both their difficulty and the range of challenging tracking scenarios that they represent (Kristan et al., 2015). The VOT challenge uses a reset-based methodology, meaning that whenever bounding box with zero overlap with the ground-truth is predicted by the tracker under test, the tracker is said to have failed. It is subsequently reinitialised five frames after failure.

Tracking performance is evaluated using the VOT's three main criteria, average overlap, robustness, and expected average overlap (EAO). Average overlap measures the Intersection-over-Union (IOU, Equation 2.3) between the ground-truth and predicted bounding boxes during tracking. Robustness corresponds to the number of times, on average, the tracker fails, i.e. completely loses its target, per video, during tracking. EAO is an estimator of the expected average overlap a tracker would obtain on a set of short-term sequences with identical visual attributes to those in the dataset. EAO can be considered a summary statistic and is used as the basis in decision-making for selecting optimal parameter values and feature selection in the subsequent experiments. Full details of the evaluation protocol are discussed in Section 2.2.2.

For clarity, the two trackers used as baselines in the following experiments, labelled PBTS-A and PBTS-B, are reported in Table 3.1. As the chapter describes the experiments carried out on each component of the tracker following a discussion of the component itself, it is important to note the parameters used for each tracker. Some of the tracker's characteristics are evaluated on PBTS-A and some on PBTS-B,

Tracker component	Tracker		Section
	PBTS-A	**PBTS-B**	
Patch model	Features = RGB $\omega_x = \omega_y = 7$, $P = 20$, $S_{max} = 10$		Section 3.3
Initialisation	$\rho^- = 0.8$, $\rho^+ = 1.2$, $\lambda = 0.01$, $\tau = 0.85$ Placement = Superpixel centroids, $\gamma = 0.25$ Sampling = Random		Section 5.3.3 Section 4.1
Search	$G = 1000, L = 100, W = 5$ $\pi_x = \pi_y = \pi_r = \pi_s = \mathcal{N}(\cdot, \cdot)$ $\mu_x = \mu_y = \mu_r = 0$, $\mu_s = 1$ $\sigma_x = \sigma_y = 0.15 \cdot (\tilde{w} + \tilde{h})$ $\sigma_r = \pi/16$, $\sigma_s = 0.02$		Section 3.4
Matching	$R = 20$, $b = \frac{1}{2}$	$R = 20$, $b = 1$	Section 3.3
Model Update	$\beta_s = 0.1$, $\beta_c = 0$	$\beta_s = 1.7$, $\beta_c = 0.05$	Section 4.3

Table 3.1: Parameters for each component used in the two tracker baselines, PBTS-A and PBTS-B, along with the section in which each set of parameters is discussed. The trackers differ only in their part matching and model update components.

but the particular version used is a consequence of when the experiments were done in the context of the development of the tracker. Although PBTS-A and PBTS-B are very similar, as shown by Table 3.1, PBTS-A used a simplified version of the update scheme (Section 4.3) and did not incorporate new samples into its model or adjust its feature sample's counts based on the number of pixels that matched each sample (equivalent to $\beta_c = 0$). PBTS-A used the standard Bhattacharyya Distance ($b = \frac{1}{2}$), where as PBTS-B gave more weight to better matching patches ($b = 1$).

The rest of the chapter is organised as follows. The part-based object model is detailed in Section 3.3, along with experiments evaluating the proposed method using different colour feature representations. A comparison of the model to colour histograms using different colour features, as well as varying the number of patches, the size of patches, and the number of samples required to accurately track objects, is also carried out. Lastly, the amount of weight given to better matches is investigated via the use of a modified Bhattacharyya distance.

Section 3.4 describes the scheme to search for the object in subsequent frames, along with an investigation characterising the movement of objects in the VOT2016 dataset, and the evaluation of a simple motion model. Our patch model is compared to other models for use in local optimisation, and the robustness for each of these methods is explored. The proportion of global sets of candidate patches that should

be locally optimised is evaluated, as well as an investigation into which distributions, and their parameters, should be sampled from in order to track robustly. The chapter is concluded in Section 3.5.

3.3 Object Representation

In this section, the part-based representation is described, along with the candidate patch location evaluation in Section 3.3.1. The use of different colour features in investigated in Section 3.3.2, the replacement of our colour model with standard histograms in Section 3.3.3, and differing the number of patches, their size, and maximum number of samples allowed to be in each model in Section 3.3.4. Increasing the importance of good quality matches is also investigated in Section 3.3.5.

3.3.1 Sample-Based Colour Model

An outline of the object representation is shown in Figure 3.3. The object is represented by a set of P patches, of width and height ω_x and ω_y pixels respectively, with each patch (Figure 3.3a) characterised by a model $\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_S, C_S) \}$, containing S pairs of features $\mathbf{f}_s$ and their corresponding match counts C_s (Figure 3.3d). Match counts C_s indicate how many pixels within the patch have features that match $\mathbf{f}_s$, where a pixel I_i with features $\mathbf{x}_i$ is said to match the sample if $\|\mathbf{x}_i - \mathbf{f}_s\| < R$ (Figure 3.3c). This describes a patch in feature space with hyperspheres of radius R that are centred on the samples of the model (Figure 3.3d), and can be thought of as empirical histograms of the frequency of features in the patch.

Unlike histograms that partition feature space into hypercubes (bins) with fixed centres determined by the width of the bins, this model makes no assumptions as to how the feature space should be partitioned. Instead, it uses the features themselves as the centres of their bins. Additionally, representations of histograms describe the contents of each bin, regardless of whether any mass lies within it. In contrast to this, our model uses a far sparser representation. For example, if an RGB histogram of the joint colour distribution (using 8 bins per colour dimension) were to describe a uniformly coloured patch, it would still need to use 8^3 integers. Whereas

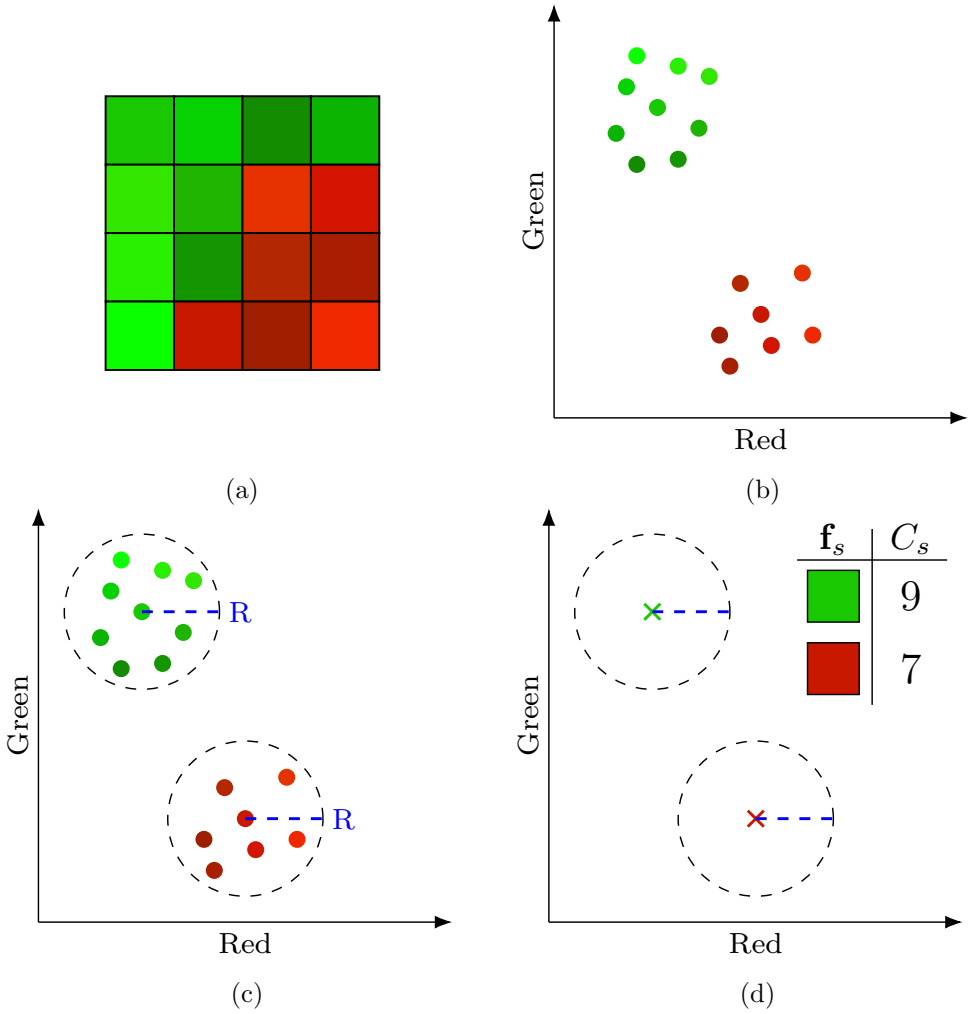


Figure 3.3: Object part model overview. The object is represented by a set of patches. A patch (a), shown in red-green colour space (b) for visualisation purposes, is characterised by spheres of feature space with radius R, centred on the pixel's features (c), and the number of samples that match them. This results in a sparse representation of the patch that is fully described by the feature-count pairs $(\mathbf{f}_s, C_s)$ and match radius R. In this example, the patch (a) is represented by two features, shown in feature space by crosses in (d), with corresponding match counts of 9 and 7.

our model only requires 4 integers per sample-count pair, limited to $S_{max} = 10$ pairs, where each pair is represented by the sample centre (3 integers) and the number of pixels that match it (1 integer). The details of how this model is initialised are given in Section 4.2.

This representation is an extension of a pixel's background model the ViBe (Barnich and Van Droogenbroeck, 2011) background subtraction algorithm. They create a model containing features extracted from the pixel and its immediate neighbours, and label a pixel value in a future frame of video as matching the model if is closer than R to a sufficient number of the modelled features. We have extended

this idea by representing image regions, in this case a patch, by feature samples and the number pixels whose features match them. Rather than defining a pixel to match its model if it it is close enough to a sufficient number of them, we count the number of pixels that match the modelled features and compare these counts to those in the model, treating them as an empirical histogram.

Given a candidate location $\tilde{\mathbf{c}}$ for a patch in an image, the generation of which is discussed in Section 3.4, its quality can be evaluated by comparing how well the features of the patch centred on $\tilde{\mathbf{c}}$ match $\mathcal{M}_p$ (Algorithm 1). This compares the features of each pixel in the patch defined by $\tilde{\mathbf{c}}$, to the features in the patch's model $\mathcal{M}_p$, counting those that match. In the case of a pixel matching multiple features in $\mathcal{M}_p$, as is the case if two or more of the patch model's features lie within $2R$ of one another, the feature that is closest to the pixel's is marked as being the one matched.

Let $\mathbf{q}$ be the normalised vector of counts corresponding to $\mathcal{M}_p$ (Algorithm 1, line 2), and $\mathbf{p}$ be the normalised vector whose j-th element corresponds to the number of pixels matched to the j-th feature in $\mathcal{M}_p$ (Algorithm 1, line 3). These two vectors/histograms, can be compared to one another using a version of the Bhattacharyya distance, proposed by Comaniciu, Ramesh and Meer (2000), that we have modified, as follows.

The Bhattacharyya distance is based on the Bhattacharyya coefficient (Bhattacharyya, 1946) between two distributions, $\mathbf{p}$ and $\mathbf{q}$, where $p_j \geq 0$, $q_j \geq 0$, $\sum p_j = \sum q_j = 1$:

$$BC(\mathbf{p}, \mathbf{q}) = \sum_{j=1}^{S} \sqrt{p_j q_j}\,. \tag{3.1}$$

Note that $0 \leq BC(\mathbf{p}, \mathbf{q}) \leq 1$ measures the overlap between two distributions, $\mathbf{p}$ and $\mathbf{q}$, with $BC(\mathbf{p}, \mathbf{q}) = 0$ if there is no overlap. We define the modified version of the Bhattacharyya Distance (MBD) to be

$$MBD(\mathbf{p}, \mathbf{q}) = (1 - BC(\mathbf{p}, \mathbf{q}))^b \tag{3.2}$$

where $b \geq 0$ controls the weight given to good matches. A larger value of b favours good matches, whereas they are down-weighted with smaller values. A parameter

Algorithm 1 Candidate patch quality evaluation.

Input: I : Image
$\tilde{\mathbf{c}}$: Candidate patch location
ω_x : Patch width
ω_y : Patch height
$\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_S, C_S) \}$: Patch model
R : Match radius
b : MBD parameter (Equation 3.2)
Output: Q : Candidate patch quality

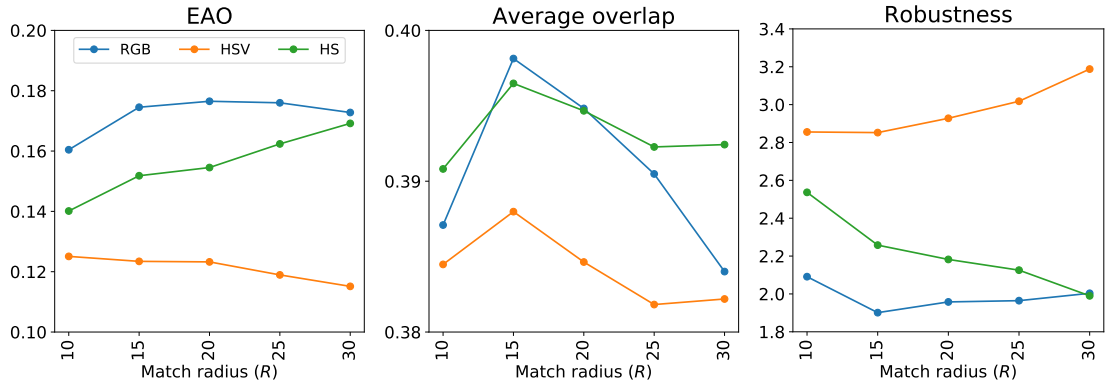
1: Set Ω to be a rectangular region of height ω_y and width ω_x, centred on $\tilde{\mathbf{c}}$
2: $\mathbf{q} \leftarrow [C_1, \ldots, C_S]$
3: $\mathbf{p} \leftarrow [0, \ldots, 0]$ $\quad |\mathbf{p}| = |\mathcal{M}_p| = S$ *and* p_j *contains number of matches to* $\mathbf{f}_j$
4: **for** $I_i \in \Omega$ **do** *For each pixel* I_i *in the patch* Ω_p*, with features* $\mathbf{x}_i$
5: $\quad \mathcal{D} = \{ \|\mathbf{x}_i - \mathbf{f}_j\| < R \mid \mathbf{f}_j \in \mathcal{M}_p \}$
6: $\quad$ **if** $\mathcal{D} \neq \varnothing$ **then**
7: $\quad\quad \mathbf{f}_j = \underset{\mathbf{f}_j \in \mathcal{M}_p}{\operatorname{argmin}} \{ \|\mathbf{x}_i - \mathbf{f}_j\| \}$
8: $\quad\quad p_j \leftarrow p_j + 1$ *Increment corresponding count*
9: $\quad$ **end if**
10: **end for**
11: $\mathbf{q} \leftarrow \mathbf{q} \,/\, |\Omega_p|$ *Normalise* p *and* q *by the patch size*
12: $\mathbf{p} \leftarrow \mathbf{p} \,/\, |\Omega_p|$
13: $MBD \leftarrow (1 - BC(\mathbf{p}, \mathbf{q}))^b$ *Modified Bhattacharyya distance (Eqn. 3.2)*
14: $Q \leftarrow 1 - MBD$ *Candidate patch similarity*

value of $b = \frac{1}{2}$ recovers the original Bhattacharyya distance as defined by Comaniciu, Ramesh and Meer (2000), which is also equal to the Hellinger distance $H(\mathbf{p}, \mathbf{q}) = \frac{1}{\sqrt{2}} \sqrt{\sum_j^S \left(\sqrt{p_j} - \sqrt{q_j}\right)^2}$ (Hellinger, 1909). Note that outside the tracking and image processing communities the Bhattacharyya distance is usually defined as $D_B(\mathbf{p}, \mathbf{q}) = -\log(BC(\mathbf{p}, \mathbf{q}))$. An investigation into the performance effects of b are shown in Section 3.3.5.

In fact, in the candidate patch, $\sum p_j \leq 1$ because only the pixels that have features matching those in $\mathcal{M}_p$ are counted. However, this is a desirable property, because when the vector is normalised by the number of pixels in a patch (Algorithm 1, line 12), it implicitly down-weights candidate patches that have fewer pixels matching the model.

We define the quality of a candidate patch location $\tilde{\mathbf{c}}$ for a given patch, modelled by $\mathcal{M}_p$, as

$$Q(\mathcal{M}_p, \tilde{\mathbf{c}}) = 1 - MBD(\mathbf{p}, \mathbf{q}), \tag{3.3}$$



(a) (b) (c)

Figure 3.4: Tracking performance using different colour spaces and match radii (R). The EAO (a), average overlap (b), and robustness (c) is shown for the RGB (blue), HSV (orange), and HS (green) colour schemes.

where $\mathbf{p}$ and $\mathbf{q}$ are defined as the empirical distributions of features for each of the patches. Therefore, the overall quality of a set of candidate patch locations $\tilde{\mathcal{C}} = \{\tilde{\mathbf{c}}_1, \ldots, \tilde{\mathbf{c}}_p, \ldots \tilde{\mathbf{c}}_P\}$, when compared to the set of patch models $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_p, \ldots \mathcal{M}_P\}$ is defined as an average of their similarity:

$$L\left(\mathcal{M}, \tilde{\mathcal{C}}\right) = \frac{1}{P} \sum_{p=1}^{P} Q\left(\mathcal{M}_p, \tilde{\mathbf{c}}_p\right) \tag{3.4}$$

3.3.2 Colour Spaces

Since it is not *a priori* obvious what the most effective colour space for tracking is (Danelljan et al., 2014a), we consider the use of three common colour spaces, RGB, HSV, and LAB. Given that the scale of each colour space is different, we also consider different match radii. The PBTS-A tracker (Table 3.1) was used in this evaluation.

We first compare the performance of RGB with HSV. Since the Hue (H) and Saturation (S) are intrinsic properties of the object, rather than the value (V), which is likely to be illumination dependent, we are motivated to also include a model (HS) with only H and S to investigate the effect of illumination. For simplicity, we rescale the HSV components to lie in the range $[0, 255]$ to match the range on which RGB components are measured. As the results in Figure 3.4 show, using RGB features with a match radius of 20 gives the highest EAO, with the second best overlap accuracy and robustness. This corresponds with qualitative results in Barnich and

Colour space	Radius				
	10	15	20	25	30
RGB	0.387 ± 0.108	0.398 ± 0.109	0.395 ± 0.104	0.390 ± 0.099	0.384 ± 0.100
HSV	0.384 ± 0.093	0.388 ± 0.094	0.385 ± 0.092	0.382 ± 0.089	0.382 ± 0.087
HS	0.391 ± 0.098	0.396 ± 0.099	0.395 ± 0.102	0.392 ± 0.102	0.392 ± 0.095

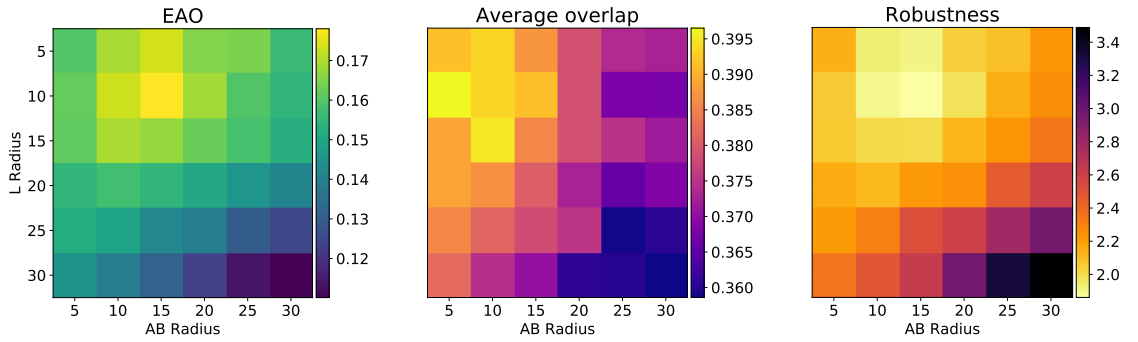
Table 3.2: Tracking performance for different colour spaces and match radii. The Average Overlap (per video) and standard deviation are shown for each combination. The first, second, and third best performances shown in their respective colours.

Colour space	Radius				
	10	15	20	25	30
RGB	2.091 ± 2.414	1.901 ± 2.216	1.958 ± 2.179	1.964 ± 2.176	2.003 ± 2.128
HSV	2.856 ± 4.091	2.852 ± 4.188	2.928 ± 4.056	3.018 ± 4.222	3.188 ± 4.505
HS	2.537 ± 3.105	2.258 ± 2.855	2.182 ± 2.808	2.126 ± 2.666	1.990 ± 2.410

Table 3.3: Tracking performance for different colour spaces and match radii. The robustness (average number of failures per video) and standard deviation are shown for each combination. The first, second, and third best performances shown in their respective colours.

Van Droogenbroeck (2011), in which they state that their "educated choice" of using a match radius of 20 (in RGB space) gives good background subtraction results. The results in Table 3.2 show that, given the differences between the average overlap for each radius and the size of their standard deviations, there is no significant difference between the schemes in terms of accuracy. However, as both Table 3.3 and Figure 3.4c show, the differences between the robustness is far more pronounced and therefore contributes much more towards the larger EAO of RGB. Although there is comparable robustness between RGB and HS, the former is the more consistent of the two colour schemes as its standard deviations are considerably lower. Note that the size of the standard deviations of the failure rates are large. This is due to the number of videos in which the tracker does not fail on and fails multiple times on, resulting in a skewed distribution of failure rates. However, while this means that the values themselves are less important on their own, comparing between two tracker's standard deviations still shows their comparative performance.

It is interesting to note the difference between HSV and HS, as HS is far less robust than HSV. In the HSV colour model, decreasing the V (value) channel corresponds to increasing blackness, while decreasing the S (saturation) channel corresponds to increasing the whiteness (Smith, 1978). This indicates that including information to distinguish between pixels under different lighting conditions appears



(a) (b) (c)

Figure 3.5: Tracking performance using the LAB colour space with different match radii for the L and A-B dimensions of LAB space. The EAO (a), average overlap (b), and robustness (c) is shown for different match radii. Note that a lighter colour (more yellow) corresponds to higher performance for each metric. The optimal pair of radii for L and AB, 10 and 15 respectively, has an EAO of 0.178, accuracy of 0.391 ± 0.116, and robustness of 1.891 ± 2.133.

to be detrimental to this colour model. We also note that, somewhat surprisingly, it is HS and RGB that achieve similar performance (Figure 3.4), rather than RGB and HSV. It might be expected that RGB and HSV would perform more similarly as they both contain the same information (encoded differently), whereas HS discards the intensity information.

We also investigate the use of the LAB feature space. Since there is a non-linear transform from RGB to LAB (Section 2.1.1), the RGB colour cube maps onto the interior of a cuboid with corners at $[0, -86.183, -107.857]$ and $[100, 98.233, 94.478]$ in LAB colour space. Note that the usable range of L is roughly half that of A and B. We therefore treat the lightness (L) channel separately from the two colour channels and use a different match radius for it; resulting in a matching ellipse rather than a matching sphere.

Figure 3.5 shows the change in performance corresponding to different radii in the L and AB dimensions. Note that for each of the three figures, a lighter (more yellow) colour corresponds to a more desirable performance in the corresponding evaluation metric. As can be seen from the three figures, using a smaller match radius for the lightness dimension is preferable in the majority of radii combinations and that the region of highest performance occurs in the same place for all three measures. It is interesting to note that the match radius for the lightness component (L) is relatively larger than that of the colour components (A-B), meaning pixels that

are illuminated differently, but that have similar colours, are more likely to match. This appears to be similar to the advantages of using HS over HSV.

Comparing the performance of the tracker using the optimal radii for LAB and RGB colour spaces (10 for L, 15 for AB, and 20 for RGB), shows that there is little difference between the two spaces, despite the non-linear transform between them. We surmise that this is because matches are occurring at a local level, where the mapping effects are roughly linear, and so the global non-linear transformation has little effect. Both LAB and RGB have roughly the maximum EAO of 0.178 and 0.177, with average overlaps of 0.398 ± 0.109 and 0.391 ± 0.116, and a robustness of 1.958 ± 2.179 and 1.891 ± 2.133. This demonstrates that the patch model performs equally well with differing colour spaces. In addition, the colour matching process is robust to changes in match radius because the performance degrades smoothly as the size of the deviation from the optimal radius increases. We therefore choose to use the RGB colour scheme with a match radius of 20 for the simplicity of not having the computational overhead of converting RGB images to LAB space.

3.3.3 Template Matching

In this section, we replace the patch model in PBTS-B (Table 3.1) with a typical template-based representation, common in a variety of part-based trackers (Battistone, Petrosino and Santopietro, 2017; Bertinetto et al., 2016a; Čehovin, Leonardis and Kristan, 2016a; Čehovin, Kristan and Leonardis, 2013; Hare et al., 2016; Xiao, Stolkin and Leonardis, 2015; Yi et al., 2015). This will allow the evaluation of our patch model's contribution towards tracking performance. The use of three colour spaces, RGB, HSV, and LAB, for the template model is evaluated along with using different numbers of partitions in each feature dimension and different update rates.

In this comparative model, patches are represented by templates (feature vectors) consisting of concatenated histograms of each independent colour channel, e.g. R, G, and B for RGB. Use of the histogram of the joint distribution, rather than the marginal distributions, of colours was computationally prohibitive as they would need to be extracted individually for each of the (70000) candidate patch

locations evaluated in the global and local searches. In contrast, calculation of the independent colour channel histograms can be carried out much faster via the use of the integral histogram (Porikli, 2005).

The RGB and HSV colour spaces are divided equally along each colour dimension into their respective partitions. LAB space is divided up such that the L dimension has half as many partitions as the A and B dimensions. This is carried out because, as noted in Section 3.3.2, the L dimension has approximately half the usable length of A and B. Each patch template $T_p(t)$ is updated via a linear combination of the previous template $T_p(t-1)$ and the template at the best estimate of the patch location $T_p^*(t)$:

$$T_p\left(t\right) = \left(1-\alpha\right) T_p\left(t-1\right) + \alpha T_p^*\left(t\right). \tag{3.5}$$

Figure 3.6 shows the performance for different numbers of bins used for each of the three colour features compared with our tracker (dashed line). Note that in Figure 3.6c additional evaluations were carried out for LAB features using additional partitions and increased update rates. They were carried out as the performance of the tracker continued to increase as both the density of bins and template update rate increased. Both RGB and HSV features gave approximately the same overall tracking performance (EAO) for each of the partition schemes (Figures 3.6a and 3.6b), whereas the performance for LAB increased as the number of partitions increased, peaking at 16 partitions of the L dimension and 32 of the A-B dimensions (Figure 3.6c).

The speed at which each of the three template models is updated appears to be negatively correlated with accuracy (second row of Figure 3.6), with smaller update rates gaining higher average accuracy scores. However, as the third row of the figure shows, that once a sufficiently large update rate is used, then the rate at which the tracker fails appears to stabilise. This result reflects the conflicting objectives in updating tracking models: increasing the update rate leads to better model adaptation at the cost of accuracy, as the chance of introducing spurious measurements increases.

It is interesting to note the accuracy for all three feature spaces is roughly 5% better than that of our part-model, where as the robustness is roughly 50% and 25%

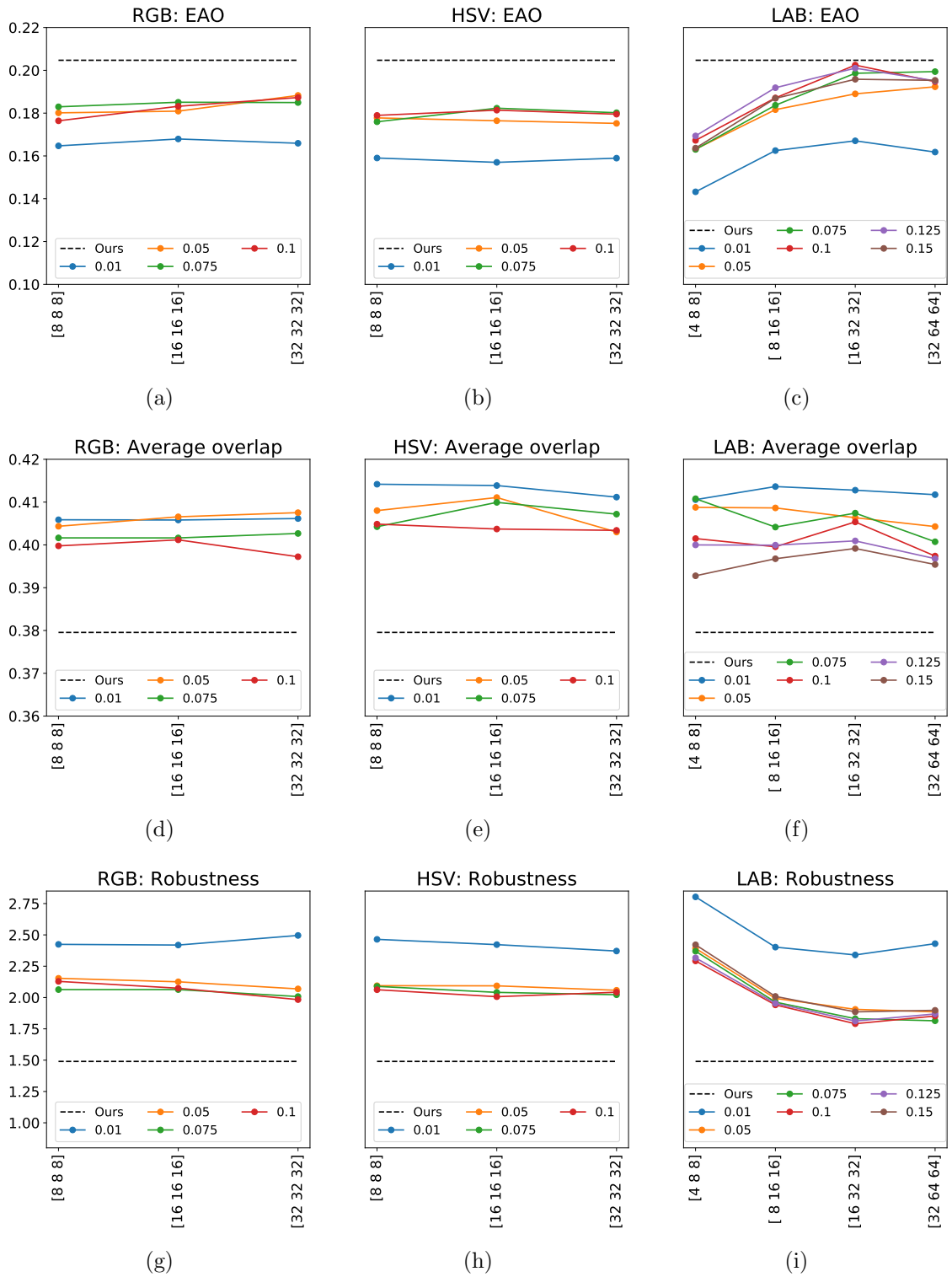


Figure 3.6: Tracking performance when replacing our colour model with a standard template-matching scheme using different colour spaces. The EAO (a-c), average overlap (d-f), and robustness (g-i) is shown for the RGB (a,d,g), HSV (b,e,h), and LAB (c,f,i) feature spaces using different template update rates (colours) and binning schemes (x-axis). Note the x-axis corresponds to the number of bins in each feature dimension, e.g. RGB using an [8 8 8] binning scheme corresponds to using 8 bins for the R, G, and B dimensions of RGB space.

worse for RGB/HSV and LAB respectively. This shows that while our patch-model is less accurate when it comes to estimating the object's bounding box, it is better at tracking the object over longer periods of time. We speculate this may be due to the model update scheme, as, like the template method, we update the match (bin) counts, but, unlike the template method, we also update the location of the bin centres. In our model, if the majority of pixels matched to a feature sample are not centred on the sample, the sample's location is interpolated towards (and in fact past as we use a predictive update rate of $\beta_s = 1.7$) the mean of the matched pixels. This results in the model's bins effectively following, in feature space, the pixels that match to the patch. This is in stark contrast to the histogram model, where features that once matched a bin would fall into another bin, resulting in a lower match quality.

3.3.4 Patch Number, Size, and Number of Samples

In the model-free tracking scenario it is not known in advance what size the tracked object is, and, as shown previously in Figure 2.1a, videos, as well as the objects they contain, can be various widths and heights. Therefore, using PBTS-A (Table 3.1), we investigate changing both the size of patch and also the number of them used to track an object. PBTS-A limits the number of samples used to represent a patch, keeping the S_{max} features with the highest counts. We therefore also evaluate what effect increasing the value of S_{max}, allowing for a more complex representation, has on tracking performance.

Figure 3.7 shows the performance of the tracker with differing size and number of patches, as well as for three different values of S_{max}. As can be seen from the top row in the figure, the EAO barely varies between each of the three values of S_{max}, with the top two combinations, 20 patches of size 7×7 and 35 patches of size 5×5, achieving approximately the same EAO for both 10 and 20 samples, and marginally less for 30. The lack of improvement, when increasing S_{max}, across all three measures for smaller patches may appear counter-intuitive, as one might expect that increasing the model's representational capacity would increase its tracking

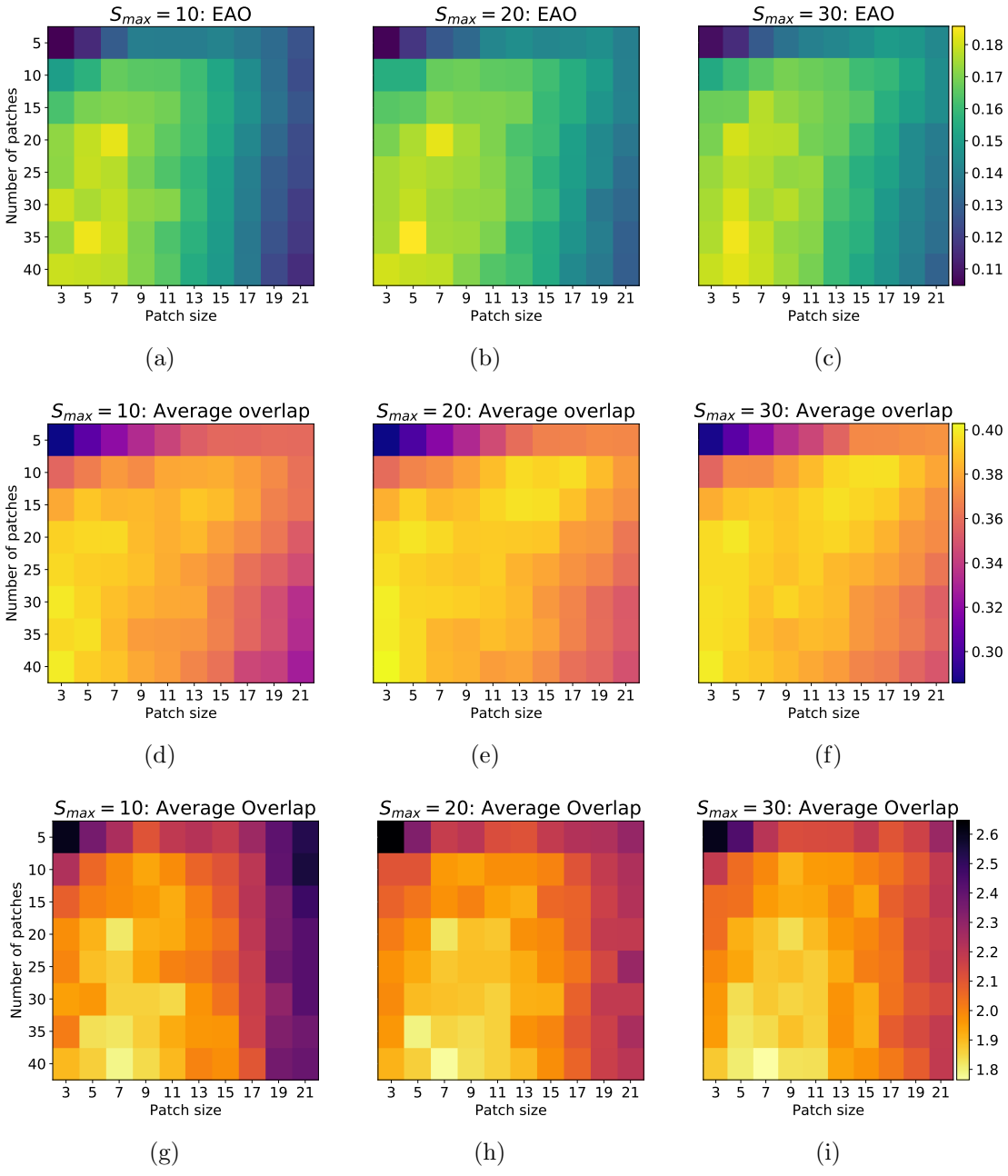


Figure 3.7: Tracking performance using different sizes (x-axis) and numbers of patches (y-axis) as well as altering the largest number of samples allowed in each patch model (columns). The EAO (a-c), average overlap (d-f), and robustness (g-i) for $S_{max} = 10$ (a,d,g), $S_{max} = 20$ (b,e,h), and $S_{max} = 30$ (c,f,i) are shown, with brighter (more yellow) colours reflecting better performance for the respective performance metric.

capabilities. However, this can be explained by the homogeneity in the modelled regions. During the initial patch placement (Section 4.2), patches are placed at the centre of superpixels, meaning that the pixels in the patch should be fairly homogeneous and therefore can be described to a sufficient standard with fewer than 30, or even 20 samples.

Conversely, for larger patches, increasing S_{max} does have a noticeable improvement in performance across all three measures (rows of Figure 3.7). We suspect that this is due to the size of the tracked objects, as typically the larger patch sizes would overlap into other superpixels. This would lead to the need for a larger number of feature samples needed in order to accurately model the feature distribution. Using 10 samples to model the features, 20 patches of size 7×7 and 35 patches of size 5×5 have approximately the same EAO of 0.1828 and 0.1838 respectively. Given that the total number of pixels contained a set of 35 5×5 patches is less than a set of 20 7×7 patches (875 compared with 980), the former results in less distance calculations (as they are all relative to the number of pixels in a patch) and therefore we select this configuration. Smaller patches also naturally result in a more sparse representation as there can be less possible feature samples per patch to represent it, which is an additional computational bonus.

3.3.5 Modified Bhattacharyya Distance

The quality of a set of candidate patches is given by Equation (3.4), it calculates the modified Bhattacharyya distance (MBD, Equation 3.2) between each candidate patch in the set and its corresponding patch model. The parameter $b \geq 0$ controls the distance between different values of the Bhattacharyya coefficient (BC, Equation 3.1) in Equation (3.2). This is illustrated in Figure 3.8, with larger values of b resulting in larger differences between a value of the MBD and BC for smaller values of BC. Therefore, in this section, using PBTS-A (Table 3.1), we investigate the effect of b in order to determine whether increasing the differences between smaller or larger values of BC results in a performance gain. For visualisation purposes we also vary the feature sample update rate β_s.

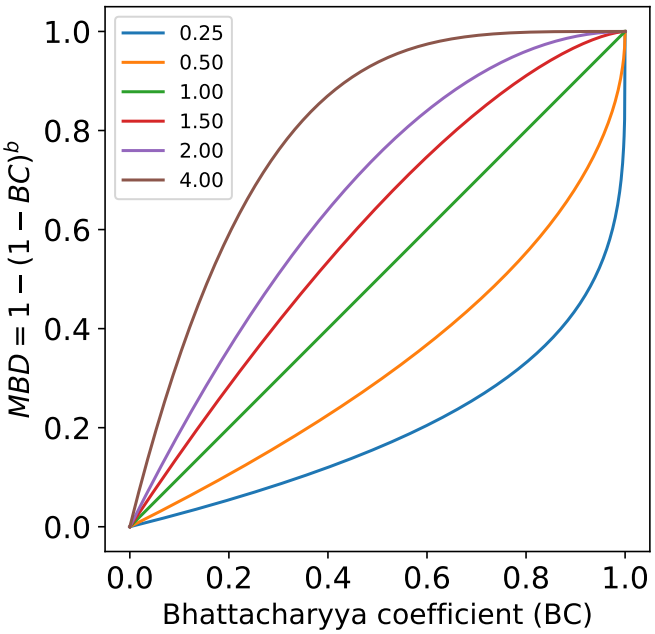


Figure 3.8: Different weightings of the Modified Bhattacharyya Distance (MBD, Equation 3.2). This shows the MBD values (y-axis) corresponding to different Bhattacharyya Coefficient (x-axis) values, for given values of b (colours). Note that $b = \frac{1}{2}$ gives the standard Bhattacharyya Distance (orange).

Figure 3.9 shows the results of this experiment. In light of the inherent noise in the performances measures, due to the stochastic nature of the tracker, Figure 3.10 shows the results of the experiment after smoothing by a convolution with a 3×3 Gaussian kernel with a standard deviation of 1. This produces a smoother version of the results and better shows how the performance of the tracker changes as both β_s and b are altered.

Figure 3.10a shows that there are significant performance benefits of increasing b beyond the default Bhattacharyya distance value of $\frac{1}{2}$. This demonstrates that not allowing the average MBD of a set of candidate patches to be dominated by a few patches with very high BC values, as would be the case with lower values of b, is preferable. Increasing the value of b does, however, decrease the tracking accuracy marginally as shown by the slight decrease in average overlap in Figure 3.10b. The tracker is more robust using larger values of b (Figure 3.10c), indicating that biasing the matching process towards weighting the good matches more highly leads to more robust tracking.

Figure 3.11 shows that there is a trade-off between average overlap and robustness, with significant increases in accuracy at the cost of increasing the risk of

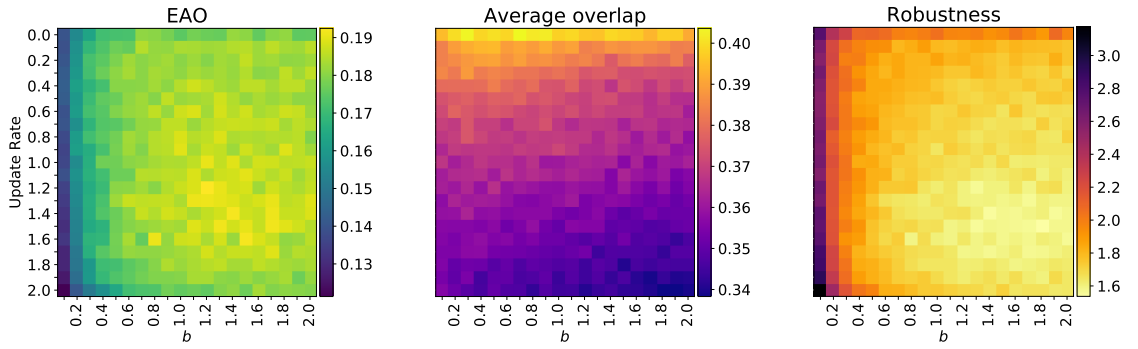


Figure 3.9: Tracking performance for different weightings of the Modified Bhattacharyya Distance (x-axis) compared with different feature sample update rates (y-axis). The EAO (a), average overlap (b), and robustness (c) for different rates are shown, with brighter (more yellow) colours reflecting better performance for the respective performance metric.

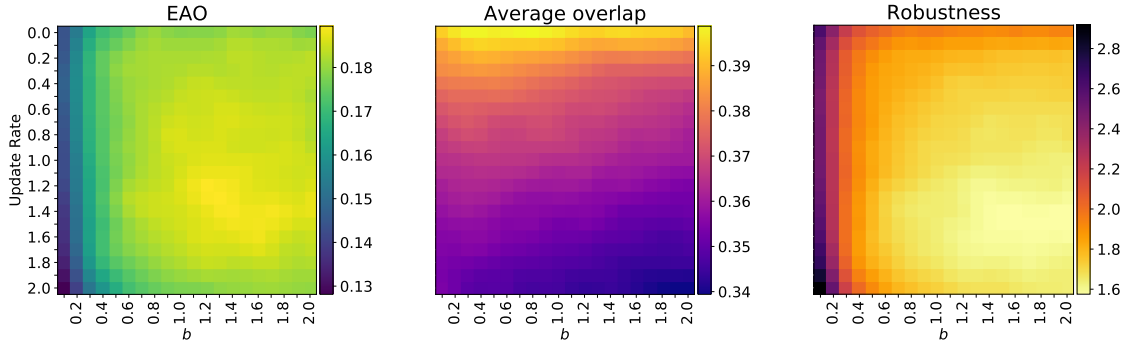


Figure 3.10: Tracking performance, convolved with a 3×3 Gaussian kernel with a standard deviation of 1, for different weightings of the Modified Bhattacharyya Distance (x-axis) compared with different feature sample update rates (y-axis). The smoothed EAO (a), average overlap (b), and robustness (c) for different rates are shown, with brighter (more yellow) colours reflecting better performance for the respective performance metric.

tracking failure. This suggests that using a tracker with multiple models for each part, one focusing on tracking accuracy and one focusing on robustness to failure, may potentially allow for a scenario in which an increase in both tracker robustness and accuracy is possible. In Figure 3.10a it is clear that there is no one optimal value for b, but rather a range of optimal values that lie somewhere around $b \in [1.0, 1.8]$. We choose to use $b = 1.4$ for the tracker as this value lies in the middle of this region of high performance.

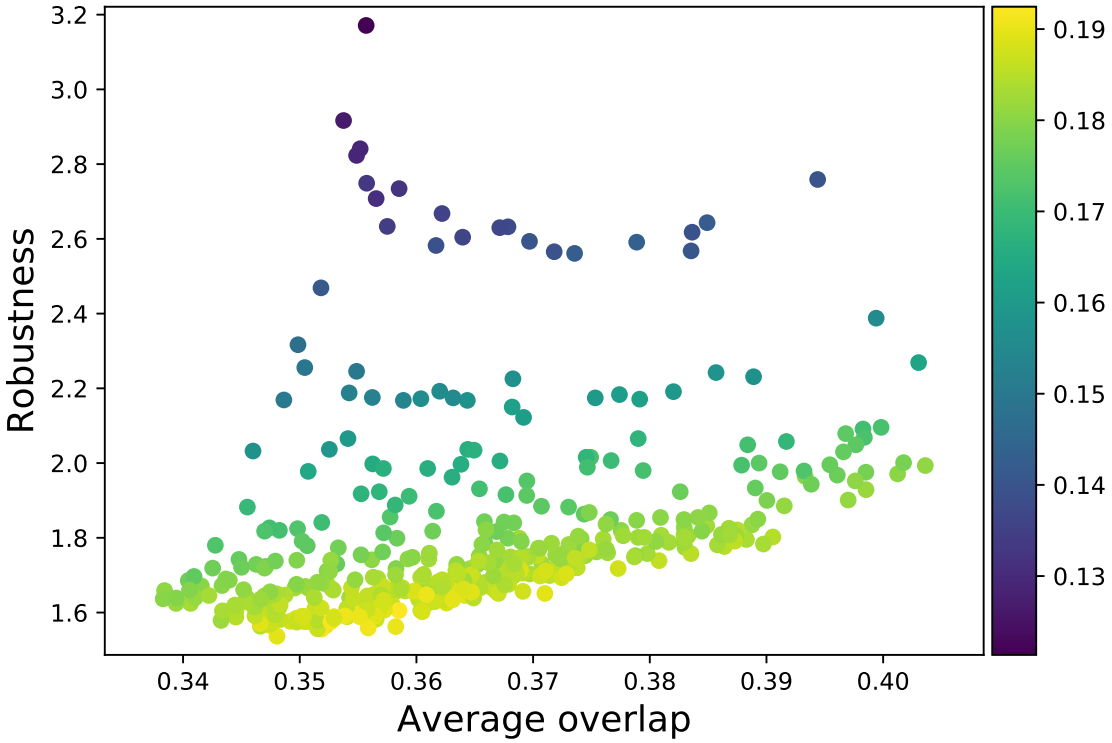


Figure 3.11: The EAO trade-off between average overlap (x-axis) and robustness (y-axis) for different combinations of patch weighting (b) and update rates (β_s). The colour of each point indicates its EAO value, with brighter colours corresponding to higher values of EAO.

3.3.6 Summary

We have proposed a new part-based object model, a feature sample-based representation that empirically characterises an image patch with a set of feature samples and their respective counts of features in the patch. Three colour spaces have been compared for use as the feature representation, with RGB having the joint highest performance and the least computational cost. A template matching scheme, using three different colour spaces, was also compared to our representation across a range of update rates and numbers of histogram bins in each dimension. It was found that our model was far more robust than the template matching schemes, but performed slightly worse with respect to the average overlap.

The number of patches, their size, and the maximum number of samples used to represent them was also investigated. There was little performance gain in allowing a richer representation (a larger maximum number of samples), but there were large differences in performance for various sizes and numbers of patches, with larger

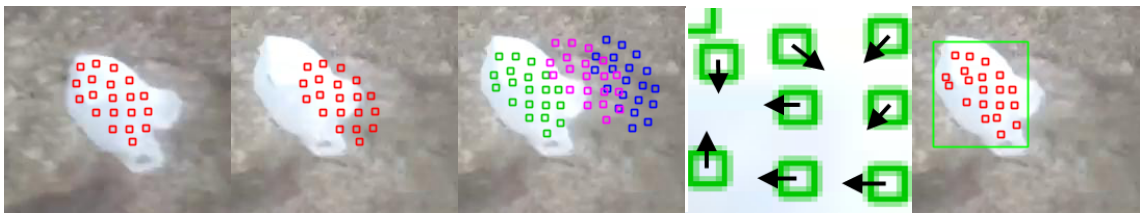
numbers of smaller patches being preferable. Lastly, the effects of increasing the weight of better or worse patches, by increasing the difference between larger or smaller values of the BC was also investigated. It was found that increasing the distance between lower values of BC, i.e. giving more weight to better matches, increased performance when compared to the standard Bhattacharyya distance ($b = \frac{1}{2}$). This increase in overall EAO came at the cost of a decrease in average overlap for the tracker, but the additional robustness far outweighed (in terms of increase in EAO) the loss of accuracy.

3.4 Object Localisation

Having described our novel patch model and investigated its characteristics, we now describe the two-part process of searching for the modelled object in subsequent frames (Section 3.4.1) by generating sets of candidate patch locations and locally optimising the best of them. The characteristics of object movement between consecutive frames of the VOT2016 dataset are investigated in Section 3.4.2, along with the use of a simple motion model in Section 3.4.3. Different methods for the local optimisation scheme and their robustness are evaluated in Section 3.4.4. The optimal number of sets of candidate patch locations to generate, as well as the the proportion of these to locally optimised is evaluated in Section 3.4.5. Lastly, several distributions from which the candidate patch locations are drawn from are evaluated in Section 3.4.6.

3.4.1 Search Scheme

In order to match our observation that object movement is largely rigid between consecutive frames and that it only has minor deviations from rigidity on the local level, we propose the following scheme. We first generate non-shearing affine transformations of the previous frame's patch locations to match the rigidity assumption, and then optimise these within a small window to allow for local non-rigidity. A schematic of the object localisation scheme is shown in Figure 3.12.



(a) (b) (c) (d) (e)

Figure 3.12: A schematic of the object localisation scheme. Given the patch locations in the previous frame (a), and based on their location in the current frame (b), we generate sets of candidate patch locations (c). These are evaluated and the best sets of patch locations are locally optimised (d), and, once re-evaluated, the best of these are use as the predicted location of the object (e).

Given the positions of the modelled patches $\mathcal{C} = \{\, \mathbf{c}_p \,\}_{p=1}^{P}$ at the previous time-step (Figure 3.12b), G sets of candidate patch locations are generated (Figure 3.12c) using Algorithm 2. A set $\mathcal{G}_p$ of candidate patch locations for the p-th patch is generated by creating a non-shearing affine transformation matrix $\mathbf{A}_g = \mathbf{TSRO}$ consisting of a translation to the origin $\mathbf{O}$, a clockwise rotation $\mathbf{R}$ by r radians, an isotropic scaling $\mathbf{S}$ by a factor s, and a translation $\mathbf{T}$ by a vector of $[x, y]^T$:

$$\mathbf{A}_g = \begin{bmatrix} 1 & 0 & c_x + x \\ 0 & 1 & c_y + y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos r & -\sin r & 0 \\ \sin r & \cos r & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$= \begin{bmatrix} s\cos r & -s\sin r & -c_x s\cos r + c_y s\sin r + c_x + x \\ s\sin r & s\cos r & -c_y s\sin r - c_y s\cos r + c_y + y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

Note that the patch locations are represented in homogeneous coordinates $\mathbf{c}_p = [c_x, c_y, 1]^T$, where (c_x, c_y) is the centroid of the patch. Each of the affine parameters are drawn from

$$x \sim \pi_x(\mu_x, \tilde{w}\sigma_x) \tag{3.8}$$

$$y \sim \pi_y(\mu_y, \tilde{h}\sigma_y) \tag{3.9}$$

$$r \sim \pi_r(\mu_r, \sigma_r) \tag{3.10}$$

$$s \sim \pi_s(\mu_s, \sigma_s), \tag{3.11}$$

Algorithm 2 Generate sets of candidate patch locations.

Input: $\mathcal{C}$: Patch centroid locations in homogeneous coordinates
G : Number of global samples to generate
$(\tilde{w}, \tilde{h})$: Predicted width and height of object
$\pi_x(\mu_x, \sigma_x)$: Distribution for x movement
$\pi_y(\mu_y, \sigma_y)$: Distribution for y movement
$\pi_r(\mu_r, \sigma_r)$: Distribution for rotation
$\pi_s(\mu_s, \sigma_s)$: Distribution for scale
Output: Sets of candidate patch locations $\mathcal{G} = \{ \Omega_1, \Omega_2, \ldots, \Omega_G \}$

1: $\mathcal{G} \leftarrow \varnothing$
2: **while** $|\mathcal{G}| < G$ **do**
3: $x \sim \pi_x(\mu_x, \tilde{w}\sigma_x)$ *Sample from parameter distributions*
4: $y \sim \pi_y(\mu_y, \tilde{h}\sigma_y)$
5: $r \sim \pi_r(\mu_r, \sigma_r)$
6: $s \sim \pi_s(\mu_s, \sigma_s)$
7: $\mathcal{G}_p \leftarrow \varnothing$
8: **for** $\mathbf{c}_p \in \mathcal{C}$ **do** *Transform each patch centroid*
9: Create affine matrix $\mathbf{A}$, centred on $\mathbf{c}_p$, with parameters (x, y, r, s) *(Equation 3.7)*
10: $\mathcal{G}_p \leftarrow \mathcal{G}_p \cup \{ \mathbf{A}\mathbf{c}_p \}$
11: **end for**
12: $\mathcal{G} \leftarrow \mathcal{G} \cup \{ \mathcal{G}_p \}$
13: **end while**

where $\pi(\mu, \sigma)$ is a distribution, typically Gaussian, with location parameter μ and scale parameter σ, and $\tilde{w}$ and $\tilde{h}$ are the predicted width and height of the object at the previous time-step. Secondly, $\mathbf{A}_g$ is applied to each patch position (Algorithm 2, lines 8-11) to generate one set of candidate locations. This process is then repeated until G sets have been created.

The distributions that the horizontal (x) and vertical (y) translations are drawn from, Equations 3.8 and 3.9, have their scale parameters scaled by the predicted width $\tilde{w}$ and height $\tilde{h}$ of the object's bounding box at the previous time-step. This allows for the predicted movement of the object to be adjusted relative to its size, because generally, larger objects move further than smaller objects, but are comparatively similar when this movement is considered as a proportion of its own size. A person walking, for example, has the same speed, relative to their size, regardless of the size of the image they are in. In Section 3.4.2 we characterise the motion of objects more precisely.

Once all G sets of candidate patches have been generated, their match quality

Algorithm 3 Locally optimise candidate patch location.

Input: I : Image
$\tilde{\mathbf{c}}$: Candidate patch location
$\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_S, C_S) \}$: Patch model
W : Local optimisation window width and height
Output: $\tilde{\mathbf{c}}^*$: Locally optimised candidate patch location

1: $\tilde{\mathbf{c}}^* \leftarrow \tilde{\mathbf{c}}$
2: $L^* \leftarrow Q(\mathcal{M}_p, \tilde{\mathbf{c}}^*)$ *Evaluate the match quality of* $\tilde{\mathbf{c}}^*$ *(Equation 3.3)*
3: Set Ω_l to be a square region with side length W, centred on $\mathbf{c}_p$
4: **for** $\tilde{\mathbf{c}}_l \in \Omega_l$ **do**
5: $L \leftarrow Q(\mathcal{M}_p, \tilde{\mathbf{c}}_l)$
6: **if** $L > L^*$ **then** *If the new location is better*
7: $L^* \leftarrow L$
8: $\tilde{\mathbf{c}}^* \leftarrow \tilde{\mathbf{c}}_l$
9: **else if** $L = L^*$ **then** *If the new location has the same quality*
10: **if** $\|\tilde{\mathbf{c}}_l - \tilde{\mathbf{c}}\| < \|\tilde{\mathbf{c}}_l - \tilde{\mathbf{c}}^*\|$ **then** *and is closer to the unoptimised location*
11: $L^* \leftarrow L$
12: $\tilde{\mathbf{c}}^* \leftarrow \tilde{\mathbf{c}}_l$
13: **else if** $\|\tilde{\mathbf{c}}_l - \tilde{\mathbf{c}}\| = \|\tilde{\mathbf{c}}_l - \tilde{\mathbf{c}}^*\|$ **then** *or is the same distance away*
14: $\tilde{\mathbf{c}}^*, L^* \leftarrow \text{choose}(\{(\tilde{\mathbf{c}}^*, L^*), (\tilde{\mathbf{c}}_l, L)\})$
15: **end if**
16: **end if**
17: **end for**

is calculated (Equation 3.4) by taking the average quality (Equation 3.3) of each transformed patch location $\tilde{\mathbf{c}}$ and its corresponding model $\mathcal{M}_p$. The location of each patch in the best L sets of these are locally optimised (Figure 3.12d) by an exhaustive search of potential patch locations within in a square window with side length W pixels, centred on the transformed patch $\tilde{\mathbf{c}}$ (Algorithm 3). The match quality (Equation 3.3) of the patch centred at each location within the window is evaluated and the location with the highest quality is selected as the patch's new predicted location $\tilde{\mathbf{c}}^*$ (Algorithm 3, lines 6-8). If there are two (or more) locations with the same match quality then the one closest to the unoptimised location $\tilde{\mathbf{c}}$ is selected (Algorithm 3, lines 10-12), and if there are multiple locations equidistant from $\tilde{\mathbf{c}}$ then we randomly select one (Algorithm 3, lines 13-14) as being the optimal patch location $\tilde{\mathbf{c}}^*$. The closest location to $\tilde{\mathbf{c}}$ is chosen to minimise non-affine movement. The set of locally optimised candidate patch locations that has the highest average quality (Equation 3.4) is selected as the predicted location of the object in the frame. Following preliminary experiments, we report the predicted bounding box of the

tracked object as being an axis-aligned bounding box 20% wider and taller than the axis-aligned bounding box that minimally encloses the patches after matching.

When performing the affine transformation, we limit the scaling to be isotropic and exclude shears from the class of transformations to be considered. This is to match our observation that object motion between consecutive frames of video is generally well represented by this subclass of affine transformations, with slight deviations from this occurring on the local level. The local optimisation allows for the deviations from rigidity to occur on a small scale, while keeping the object's motion approximately rigid when viewing the set of patches in its entirety.

It is worth noting that while the methods of Yi et al. (2015) and Čehovin, Kristan and Leonardis (2013) share some similarities with our object localisation scheme, there are several important differences. In contrast to Yi et al. (2015), who, after applying an affine transformation to their set of patches, randomly move each patch and evaluate its quality, we locally optimise within a small region around the affine transformed position. Yi et al. (2015) also limit the class of transformations to only include translation, where as we include both isotropic scaling and rotation. The localisation scheme of Čehovin, Kristan and Leonardis (2013) uses the cross-entropy method to first find the optimal affine transformation and then locally optimises each patch. Our methods differs from this as we locally optimise the L sets of patch locations with the highest quality, each set of which can have different patch locations to other sets.

Čehovin, Kristan and Leonardis (2013) start a search for the optimal affine transform by sampling from a Gaussian distribution with a covariance of $20\mathbf{I}$ for all sizes of objects, similar to Yi et al. (2015) who also sample from a Gaussian distribution with a scale parameter of 8 for both x and y translation. In contrast to this, we sample translations taken from distributions whose scale parameter is scaled by the respective width or height of the object, as we have observed that larger objects move a greater number of pixels per frame and smaller objects move less and we suggest that our model is closer to how objects move between frames. Note that an analysis of object motion in the VOT2016 dataset is performed in the

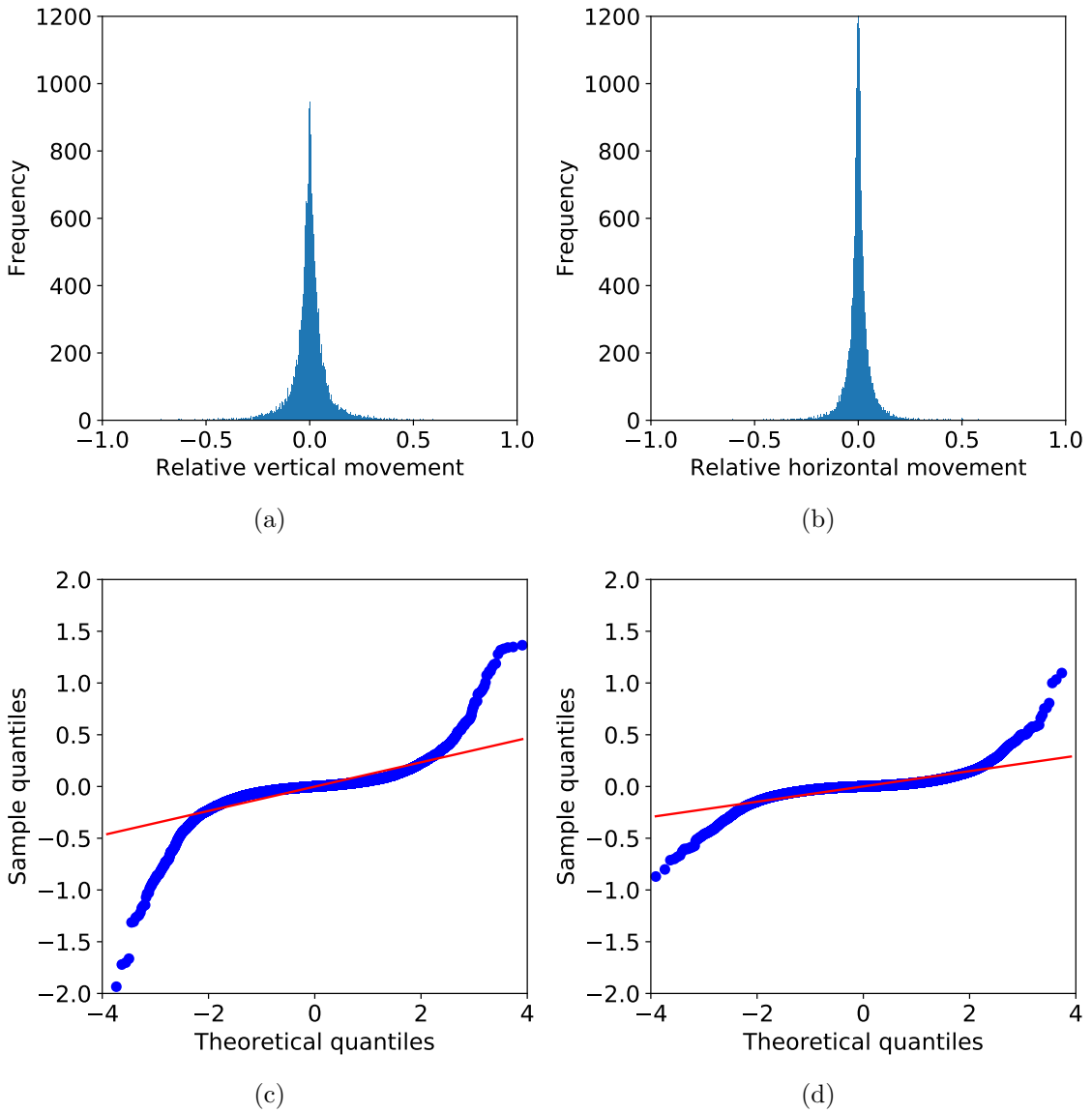
following section.

3.4.2 Object Motion in the VOT2016 Dataset

Tracking methods typically make one of two assumptions about an object's motion. Some methods assume that an object can only move in a fixed window, centred on the object, up to half its predicted width and height, such as correlation filter-based trackers (Danelljan et al., 2014a; Danelljan et al., 2016; Danelljan et al., 2015b; Henriques et al., 2015; Lukežič, Zajc and Kristan, 2018; Sun et al., 2018; Xiao, Stolkin and Leonardis, 2015). Others assume the object's motion is Gaussian in nature (Čehovin, Kristan and Leonardis, 2013; Yi et al., 2015) and may or may not include additional assumptions in the form of an explicit predictive motion model, such as a Kalman Filter (Senna, Drummond and Bastos, 2017). In this section, using the VOT2016 dataset, we analyse how much objects move, relative to their width and height. The VOT2016 dataset is designed to be representative of real-world tracking scenarios and has videos that were selected to be included in it based on their visual attributes and difficulty. We also investigate how persistence of motion can be used to predict an object's location in subsequent frames by measuring the correlation between the change in object location in a pair of frames and the change in object location in the subsequent pair of frames.

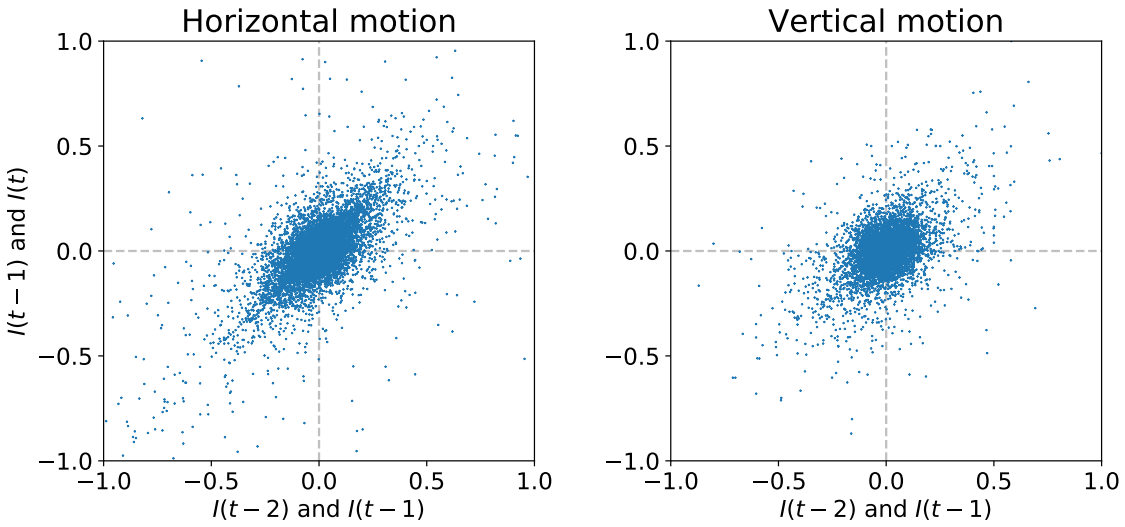
We start by comparing the centroids of an object's ground-truth bounding boxes, provided by the VOT2016 dataset, to their centroids in the previous frame. The distances between centroids in consecutive frames are calculated for both movement in the horizontal and vertical directions, and these are normalised by the width and height of the object, estimated as the dimensions of the axis-aligned bounding box fitted to the ground-truth bounding box, in the first of each pair of consecutive frames.

Figure 3.13 shows the histograms of object movement for all frames of video in the VOT2016 dataset, as well as their quantile-quantile (Q-Q) plots showing how the values are distributed when compared to a Normal distribution. As can be seen from the histograms for the relative horizontal (Figure 3.13a) and vertical



(a) (b) (c) (d)

Figure 3.13: Frequency histograms, from all pairs consecutive of frames in the VOT2016 dataset, showing the amount the tracked object moved relative to its size in the previous frame in the horizontal (a) and vertical (b) directions, as well as how these values are distributed with Q-Q plots (c and d). Note the red line corresponds to a Normal distribution with statistics scaled by the mean and standard deviation of the data.



(a) (b)

Figure 3.14: Tracked object motion in the horizontal (a) and vertical (b) direction, relative to its size in the corresponding dimension, between all pairs of consecutive frames across every video of the VOT2016 dataset.

movement (Figure 3.13b), both of these sets of movements are distributed with a mean of approximately zero. This matches the assumption made by most trackers that the most likely location for an object in the next frame is its location in the current frame. However, the shape of both distributions is clearly non-Gaussian as they are sharply peaked and heavy tailed, with an excess kurtosis[1] of 40.455 and 78.788 for the horizontal and vertical distributions respectively. This is also reflected in the Q-Q plots (Figures 3.13c and 3.13d), with the S shape indicating that the distributions are heavy tailed in comparison to the Normal distribution. This means that objects, which usually move small amounts (relative to their size), move larger distances more frequently than can be modelled with a Gaussian distribution with the mean and standard deviation of the data.

We next look at the distance an object travels, relative to its size in the previous frame, between two pairs of consecutive frames, $I(t-1)$ and $I(t)$, and $I(t-2)$ and $I(t-1)$. Figure 3.14 shows the relative amounts of horizontal and vertical motion between consecutive pairs of frames, for all videos in the VOT2016 dataset. There is a stronger correlation between the two pairs of frames for horizontal movement than vertical with correlations of 0.613 and 0.431 respectively. This indicates that

[1] The excess kurtosis of a Gaussian distribution would be 0.

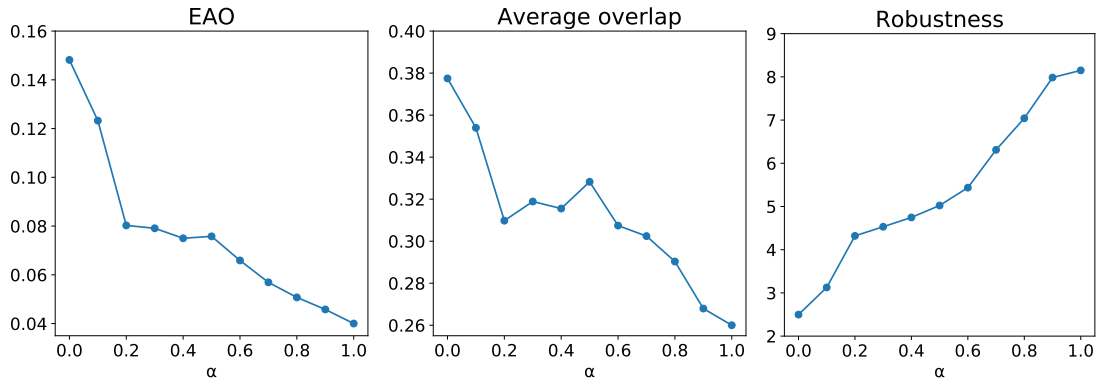
motion is more likely to persist, and therefore be more predictable, in the horizontal direction. This may be explained by many of the videos in the dataset having objects in them that travel horizontally, as motion is often primarily in the horizontal plane, and fewer videos have flying objects or those filmed from a viewpoint above the object.

However, there are plenty of occasions when there is a negative correlation (upper-left and lower-right quadrants of Figures 3.14a and 3.14b). This means that using an object's motion in the previous frame to predict its subsequent motion may sometimes be detrimental to the tracking process. A tracker might, for example, predict an object moved 25% of its height upwards, matching its motion in a the last tracked frame, when in reality it moved 25% downwards; potentially causing a tracking failure. With this in mind, the effects of using the object's motion in the previous frame to predict the object's new location on tracking performance is investigated in the next section.

3.4.3 Motion Prediction

In this section, using the tracker PBTS-A (Table 3.1) with no model update scheme ($\beta_s = 0$), we investigate using the object's motion in the previous frame of video to predict where the object is located in the current frame. This is motivated by the result in Section 3.4.2 that showed a positive correlation between the motion in both the horizontal and vertical directions between frames $I(t-1)$ and $I(t)$, and $I(t-2)$ and $I(t-1)$.

In the tracker, the parameters of the affine transformations used to generate sets of candidate patch locations (Section 3.4.1) are drawn from Gaussian distributions. These are $\pi_x(\mu_x, \tilde{w}\sigma_x)$, $\pi_y(\mu_y, \tilde{h}\sigma_y)$, $\pi_r(\mu_r, \sigma_r)$, and $\pi_s(\mu_s, \sigma_s)$, for the horizontal (x) translation, vertical (y) translation, scale, and rotational components of the transformation. The mean values of each distribution correspond to predicting the object is in the same location and size as in the previous frame, i.e. $\mu_x = 0$, $\mu_y = 0$, $\mu_r = 0$, $\mu_s = 1$. We linearly interpolate between these values and the value of the parameter corresponding to the affine transformation that gave the best match



(a) (b) (c)

Figure 3.15: Tracking performance using motion prediction (α). The EAO (a), average overlap (b), and robustness (c) for using more (larger values of α) or less (smaller values of α) motion information in the sampling scheme. A value of $\alpha = 0$ corresponds to not using any predictive information and generating affine transformations centre on the object's previous location. Conversely, $\alpha = 1$ corresponds to generating affine transformations centred on the predicted location, based on the object's motion in the previous frame.

quality in the previous frame

$$\mu_x(t) = (1 - \alpha)\mu_x + \alpha\mu_x^*(t - 1), \tag{3.12}$$

where $\mu_x = 0$ and $\mu_x^*(t-1)$ was the optimal horizontal translation from the previous frame, and $\alpha \in [0, 1]$ controls how aggressive the predictor is.

Figure 3.15 shows the results of these experiments, varying α from 0, which corresponds to sampling around the object's previous position, and 1, which means that samples are centred on the object's predicted location, rotation, and scaling. Predicting any amount of motion, based on the previous frame's motion, gives worse tracking performance in this case, with EAO (Figure 3.15a) decreasing as α increases. Both the average overlap (Figure 3.15b) and robustness (Figure 3.15c) also deteriorates as α increases. We suggest this is due to the benefits of being able to predict where the object is going to be is outweighed by the tracking failures incurred when the object moves in the opposite way to its previous motion. This is reflected by the increased failure rate as α increases (Figure 3.15c).

It might be possible to counteract the increase in failure rate by increasing the standard deviations controlling the size of the search region, σ_x, σ_y, σ_r, and σ_s. This would have the effect of searching a larger region of affine parameters, meaning that

parameters that cover a larger reversal in object motion may be included. However, our object localisation scheme (Section 3.4.1) sufficiently covers the area of object motion without the need for a motion model, as shown by the results with $\alpha = 0$.

We suggest that another contributing factor to motion models being detrimental to the tracking process is that, in the unconstrained settings of the model-free tracking, the classes of objects being tracked varies greatly. Mechanical objects, such as planes, helicopters, and cars, tend to move with a more constant velocity model, where as natural objects, such as people and animals tend to react to stimuli around them and, as they are generally travelling much slower, are able to abruptly change direction.

3.4.4 Alternative Local Optimisation Methods

In this section, using the tracker PBTS-A (Table 3.1) with no model update scheme ($\beta_s = 0$), we investigate the use the of normalised cross-correlation (NCC) in the local optimisation stage of the object localisation. Specifically, we replace the object model (Section 3.3) and comparisons between each potential new patch location during local optimisation (Algorithm 3). They are replaced by using a template of each patch, taken from the first frame of video, and comparing this to each potential new patch location using the NCC as a measure of match quality. The NCC between two identically sized image intensity patches, Ω_a and Ω_b, whose values have been standardised (patch mean subtracted and then divided by the patch standard deviation), can be calculated as

$$NCC\left(\Omega_a, \Omega_b\right) = \frac{1}{n} \sum_{\substack{I_i \in \Omega_a \\ J_i \in \Omega_b}} I_i \cdot J_i, \tag{3.13}$$

where $n = |\Omega_a| = |\Omega_b|$. This can be thought of, if the patches are represented as vectors, as the cosine of the angle between the two vectors.

We compare our patch model with a model using the NCC for image intensity (NCC-G), and, as it is unclear as to the best way to calculate the NCC for multi-dimensional features, also three different methods for calculating the NCC for colour image patches. In first method (NCC-C) we concatenate the three RGB channels

Measure	Local optimisation method				
	Ours	NCC-G	NCC-C	NCC-C-I	NCC-C-S
EAO	0.148	0.133	0.129	0.131	0.133
Average overlap	0.377 ± 0.097	0.350 ± 0.112	0.343 ± 0.109	0.345 ± 0.109	0.344 ± 0.107
Robustness	2.498 ± 2.443	2.649 ± 2.347	2.681 ± 2.471	2.626 ± 2.452	2.640 ± 2.376

Table 3.4: Tracking performance of each of the local optimisation methods. We evaluated greedily selecting the best match to our model (ours), NCC using patch intensity (NCC-G), NCC using RGB and standardising by the entire patch's statistics (NCC-C), standardising each channel separately (NCC-C-I), and treating each colour channel separately like NCC-G and averaging their NCC (NCC-C-S). The first, second, and third best performances shown in their respective colours.

together and standardise them, i.e. subtract the mean and divide by the standard deviation, by the mean and standard deviation of the entire vector. The second method (NCC-C-I) each channel is standardised separately and then the channels are concatenated together. Thirdly, NCC-C-S calculates the NCC identically to NCC-G for each of the three channels and takes the average of the three values.

The results of the comparisons can be seen in Table 3.4. Our model achieves the highest EAO as well as the highest average overlap and robustness for the dataset. It is interesting to note that there is little difference between the performance of the using NCC with the image intensity values (NCC-G) and the three variants of the RGB method. This indicates that the choice of feature in the NCC methods is not what is resulting in the poorer performance. We suggest that the lack of update scheme is more detrimental to the NCC model when compared with our own representation.

In order to further investigate the robustness of each local optimisation scheme, as well as to counteract the use of no update scheme, we compare the average IOU (Equation 2.3) of the predicted bounding box of each method when used to track short sequences of video. We extract 10 sequences of 5, 10, 20, and 30 frames in length, from each video of the VOT2016 dataset. The tracker is initialised on the first frame of each sequence and ran for its length, with the Intersection-over-Union (IOU) of its predicted bounding box measured for the final frame in the sequence. The average of this is taken over 10 independent runs of the tracker.

Table 3.5 shows the average IOU of each final frame for a sequence length, averaged across all runs and videos. We note that there is little significant difference

Frames	Local optimisation method				
	Ours	NCC-G	NCC-C	NCC-C-I	NCC-C-S
5	0.472	0.470	0.475	0.464	0.465
10	0.429	0.432	0.435	0.414	0.421
20	0.364	0.357	0.370	0.338	0.343
30	0.345	0.325	0.343	0.300	0.293

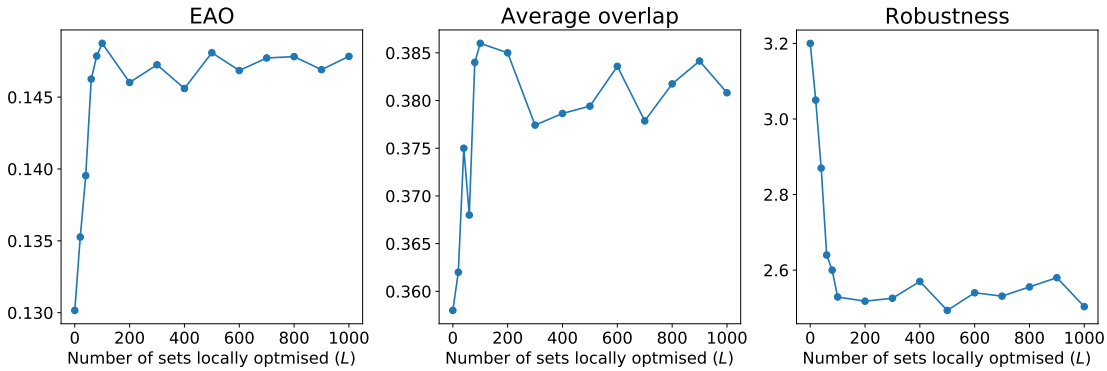
Table 3.5: Local optimisation method robustness. The average IOU is reported for each local optimisation technique, averaging the IOU taken from the last frame of sequences of lengths 5, 10, 20, and 30. This was carried out for 10 sequences of each length taken from each of the 60 videos in the VOT2016 dataset and averaged over 10 tracker runs. The first, second, and third best performances shown in their respective colours.

between our method and using NCC of both the intensity of the patches (NCC-G) and using the NCC of the RGB channels concatenated together (NCC-C). We choose to use our model in the local optimisation scheme as using an NCC-based method would require the use of an additional patch model. NCC, as the experiments have shown, would also require an update scheme as it is less robust than our method over entire videos, and therefore would need an update scheme similar to Equation (3.5).

3.4.5 Rate of Local Optimisation

Here, we investigate the effect of varying the number L of sets of candidate of patches to be locally optimised has on the performance of the tracker. Using PBTS-A (Table 3.1), with no model update ($\beta_s = 0$), we evaluate the tracker on the VOT2016 benchmark using $L = \{\, 0, 10, \ldots, 100, 200, \ldots, 1000 \,\}$, ranging from performing no local optimisation ($L = 0$) to locally optimising every set of candidate patch locations generated ($L = G = 1000$).

As shown in Figure 3.16 it is beneficial to locally optimise the most promising sets of candidate patches, when compared to not performing any local optimisation ($L = 0$). All three performance measure, follow roughly the same pattern, with performance rapidly increasing once local optimisation starts to be carried out $L > 0$. It is interesting to note that locally optimising only 10% ($L = 100$) of the sets of candidate patches provides roughly a 40% gain in EAO (Figure 3.16a), and that performance then rapidly plateaus. This suggests that the object part model is sufficiently discriminating between sets of patches with good and bad match qualities.



(a) (b) (c)

Figure 3.16: Tracking performance using different numbers (L) of best sets of candidate patches to locally optimise. The EAO (a), average overlap (b), and robustness is shown for various numbers of sets of patches to optimise (x-axis).

This is indicated by there being no significant increase in tracking performance past $L = 100$, meaning that the best sets of patches are likely to be located within the best 100 sets of patches.

3.4.6 Affine Sampling Distributions

Section 3.4.2 has shown that an object's frame-to-frame translation is distributed more like a Laplace distribution than a Gaussian (Figure 3.13). In light of this, using the PBTS-B tracker, we compare the performance of the tracker when sampling from Gaussian and Laplace distributions for the object's movement. As the tails of the distribution of horizontal and vertical movement are heavy (Figures 3.13c and 3.13d), we also look at using Latin Hypercube Sampling (McKay, Beckman and Conover, 1979) to generate samples that are more evenly distributed across affine parameter space in efforts to further capture outlier movement.

The probability density function (PDF) of the Laplace distribution is defined as

$$f\left(z \mid \mu, \sigma\right) = \frac{1}{2\sigma} \exp\left(-\frac{|z-\mu|}{\sigma}\right), \tag{3.14}$$

where z is a random variable, and μ and σ are the location and scale parameters of the PDF. As Figure 3.17 shows, the Laplace distribution is similar to the Gaussian, but is expressed in terms of the absolute distance from the mean (as opposed to the squared distance), resulting in heavier tails than the Gaussian.

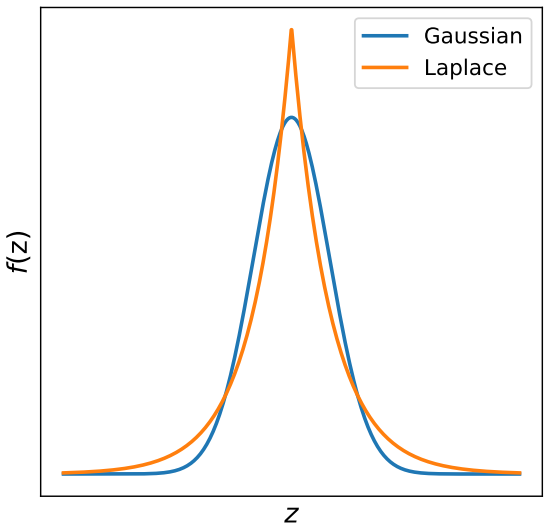


Figure 3.17: The probability density functions for Gaussian and Laplace distributions with a location parameter $\mu = 0$ and scale parameter $\sigma = 0.5$. Note that the Laplace distribution is more sharply peaked and has heavier tales than the Gaussian. This better fits the relative object movements observed in the VOT2016 dataset (Figure 3.13).

Figure 3.18 shows the performance of the tracker when drawing horizontal and vertical translation samples from a Gaussian distribution (top row) and from a Laplace distribution (bottom row). A smaller range of values of σ_x were evaluated for the Laplace distribution as its EAO (Figure 3.18a) peaked at lower values of σ_x than the Gaussian distribution's (Figure 3.18d). There is little difference in the overall best possible EAO achievable by using each of the sampling schemes, with the highest EAO having values of 0.219 and 0.217 for Laplace ($\sigma_x = 0.15, \sigma_y = 0.125$) and Gaussian ($\sigma_x = 0.25, \sigma_y = 0.2$) respectively. The best pair of parameters for the Laplace distribution has a better robustness rate (1.247) when compared with the best for the Gaussian (1.321), however, they also have a lower accuracy of 0.366 compared with the Gaussian's 0.378. It is interesting to note that, while the highest possible robustness is similar for the Gaussian and Laplace distributions (Figures 3.18c and 3.18f), the general robustness to tracking failure across a range of parameter values is much better for the Laplace distribution. This suggests that when the values of σ_x or σ_y are too small to predict large enough relative object movements using the Gaussian distribution, the increased weight in the tails of the Laplace distribution gives the tracker enough samples far enough away from the object to continue to track more robustly.

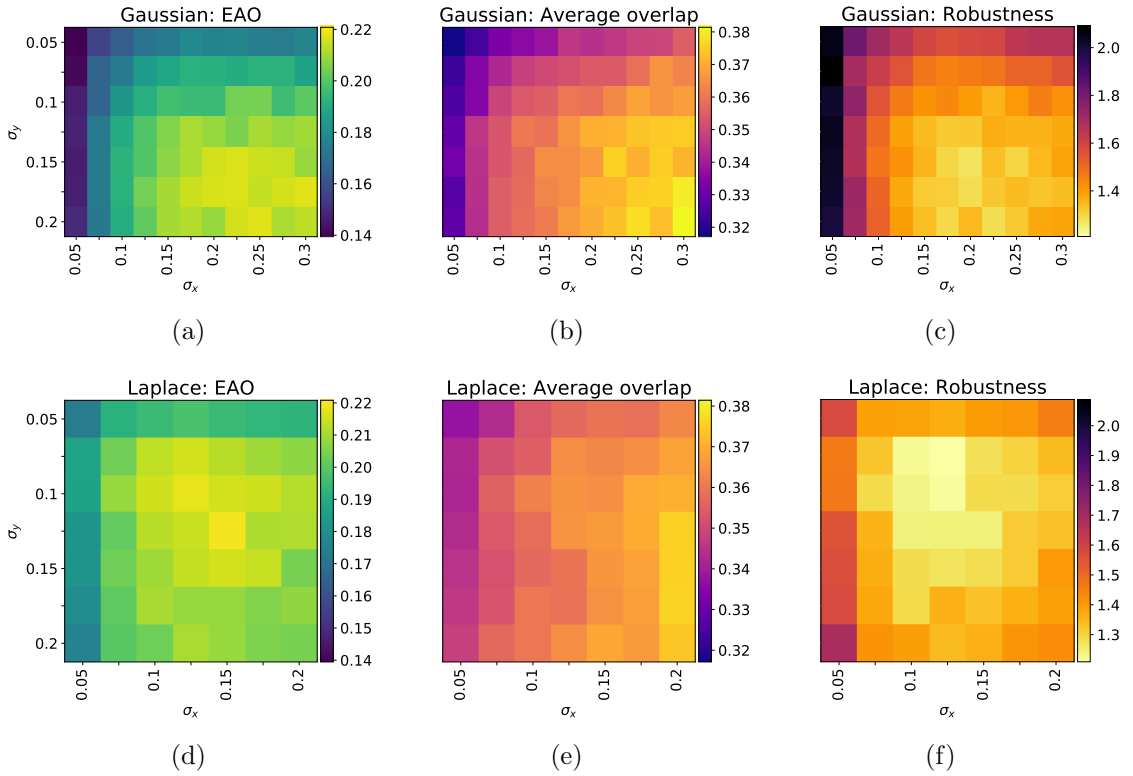
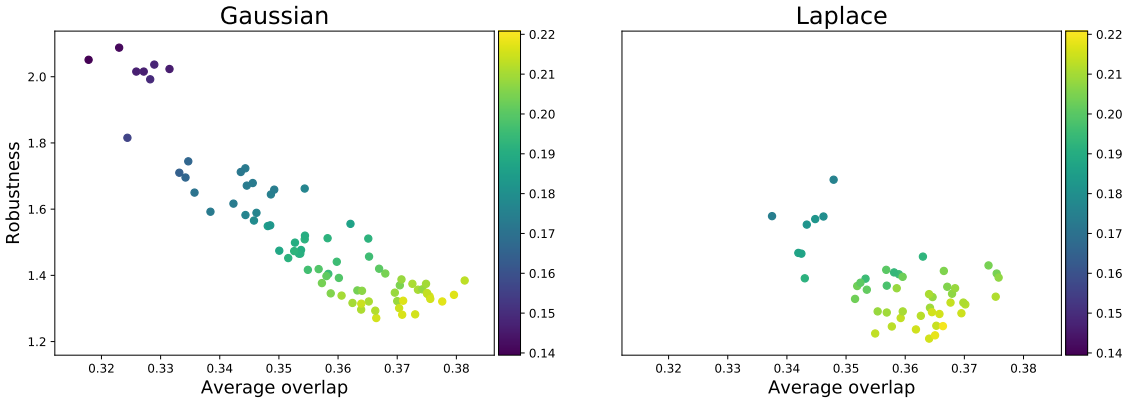


Figure 3.18: Tracking performance using different scale parameters for the Gaussian and Laplace distributions in the vertical (x-axis) and horizontal (y-axis) directions. The EAO(a,d), average overlap (b,e), and robustness (c,f) for different sizes of σ_x (x-axis) and σ_y (y-axis) are shown for the Gaussian (a-c) and Laplace (d-f) distributions. Note that a larger range of σ_x values were evaluated for the Gaussian distribution as performance did not peak in the same range as the Laplace distribution.

Similarly to the trade-off between average overlap and robustness for different values of b (Section 3.3.5), it can also be seen in Figure 3.19 that there is a trade-off between the two performance measures at the higher EAO values. We suggest that this may be due to larger values of σ_x and σ_y allowing for the tracker make sure all object parts reach objects that make large movements. Conversely, these large values may, when the object only moves small amounts, allow for the part-model to match areas of the image which are visually similar to the object but do not belong to the it and may not have been reached using smaller scaling parameters.

Next we use Latin Hypercube Sampling (McKay, Beckman and Conover, 1979) to see if the parameter space of the affine transformations are sufficiently explored when generating samples. Latin Hypercube Sampling (LHS) is a technique for generating a stratified random samples of parameter values from a multidimensional distribution, in this case 4 dimensions (i.e. x, y, r, and s). In the one dimensional case, LHS involves dividing the cumulative density function (CDF) of the distribution



(a) (b)

Figure 3.19: The EAO trade-off between average overlap (x-axis) and robustness (y-axis) for different combinations of scale parameters for the Gaussian (a) and Laplace (b) distributions. The colour of each point indicates its EAO value, with brighter colours corresponding to higher values of EAO.

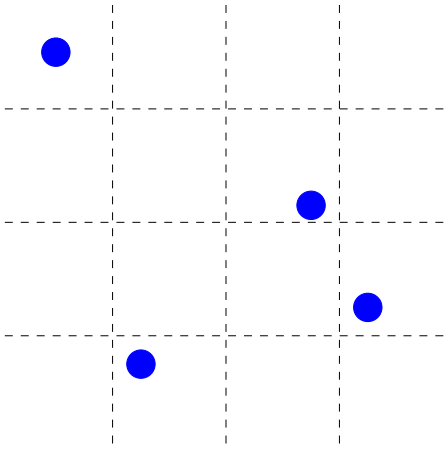
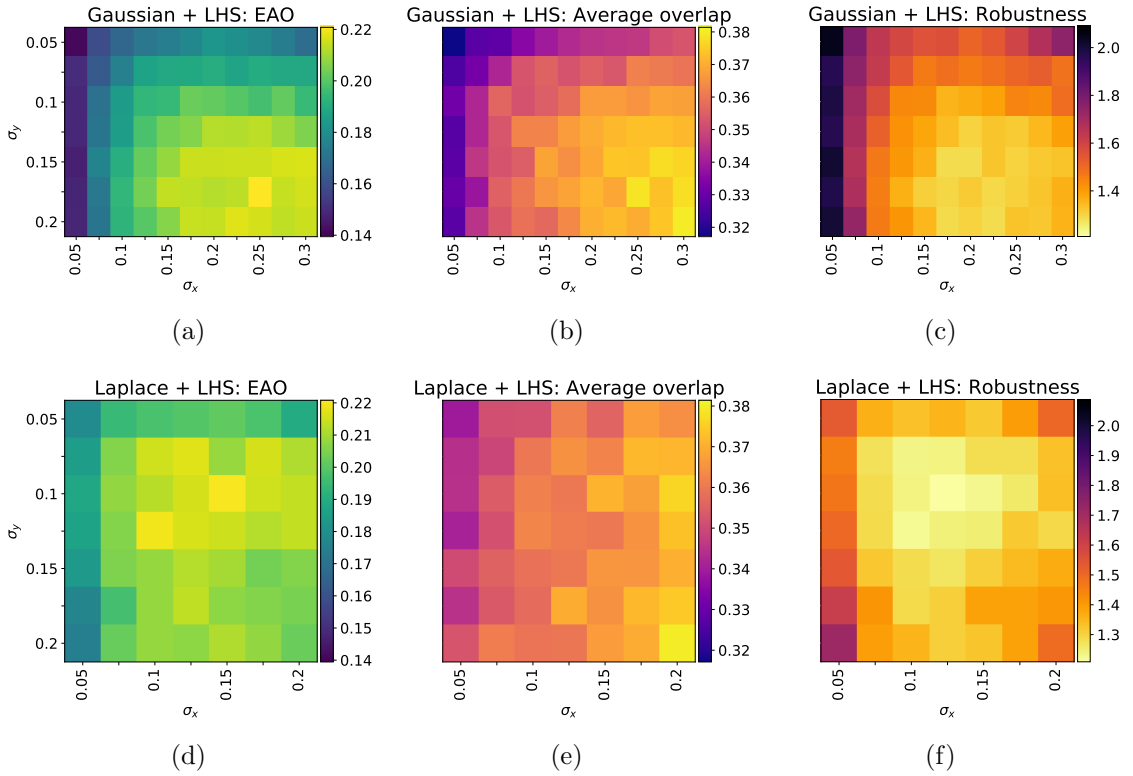


Figure 3.20: An example of Latin Hypercube Sampling in two dimensions. Each dimension is equally partitioned (dashed lines), and samples (blue) are taken randomly from each partition, ensuring one sample lies in each row and column.

to be sampled from into partitions, so that each element of the partition contains equal probability mass. A sampling point is then randomly selected within each of these partitions. This can be extended to the multidimensional case, assuming each dimension is independent of one another, by generating LHS samples for each dimension and randomly combining them such that only one combined sample lies within each row and column of the partitioned space (Figure 3.20).

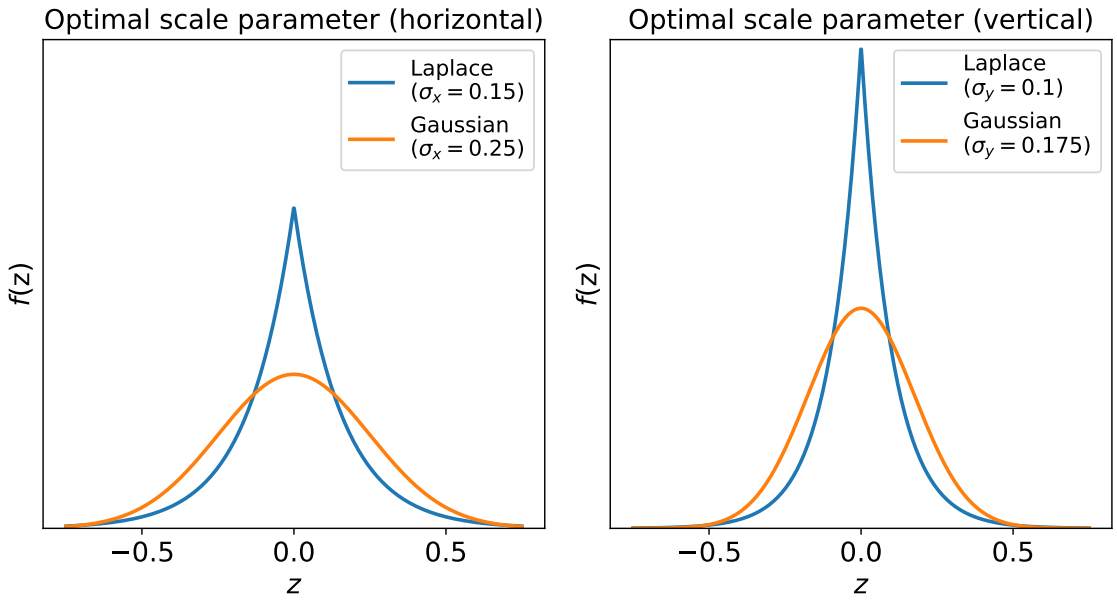
Figure 3.21 shows the performance of the tracker using LHS for the Gaussian (top row) and Laplace (bottom row) distributions. The results are very similar to those in the previous experiment (i.e. without using LHS, Figure 3.18), with only marginal improvement in performance. The highest performing pairs of parameters



(a) (b) (c)

(d) (e) (f)

Figure 3.21: Tracking performance when sampling from the Gaussian and Laplace distributions via Latin Hypercube Sampling. The EAO(a,d), average overlap (b,e), and robustness (c,f) for different sizes of vertical (σ_x, x-axis) and horizontal (σ_y, y-axis) scale parameters for the Gaussian (a-c) and Laplace (d-f) distributions. Note that a larger range of σ_x values were evaluated for the Gaussian distribution as performance did not peak in the same range as the Laplace distribution.



(a) (b)

Figure 3.22: The probability density functions of the optimal pair of Gaussian and Laplace scale parameters for horizontal (a) and vertical (b) translation. In order to sufficiently model an object's movement, the Gaussian distribution needs much larger scale parameter.

for the Gaussian and Laplace distributions are $\sigma_x = 0.25$ and $\sigma_y = 0.175$, and $\sigma_x = 0.15$ and $\sigma_y = 0.1$ respectively. These had EAO scores of 0.221 and 0.220 respectively, average overlap of 0.378 and 0.371, and robustness of 1.308 and 1.231. It is interesting to note the different sizes of optimal scale parameters for the two distributions, with Figure 3.22 showing the two distributions with their σ_y values. As can be seen from the figure, much more weight is given to larger proportional translations of the object with the optimal Gaussian distribution compared with the Laplace. We suggest that this is in order to have enough probability mass at the tails of the true distribution of object's translation.

As there is a slight improvement in performance using LHS, and that the Laplace distribution better models the true distribution of object movement, we choose to use the Laplace distribution with scale parameters of $\sigma_x = 0.15$ and $\sigma_y = 0.1$, combined with LHS.

3.4.7 Scale, Rotation, and Local Optimisation Window Size

Given that we have explored the optimal parameters for horizontal and vertical translation in the previous section, we now investigate tracking performance with respect to different scale parameters for the object scale (σ_s) and rotation (σ_r) distributions that we sample from. We also investigate the size (W) of the square window in which the greedy local optimisation takes place (Algorithm 3, Section 3.4.1).

We start by evaluating the PBTS-B tracker using scale parameter values of $\sigma_s \in \{\, 0.01, 0.02, \ldots, 0.05 \,\}$, corresponding to a 1–5% change in object size. These values approximately cover the known Interquartile Range (IQR) of scale changes for objects in the three datasets surveyed in Section 2.2.4, the largest of which, VOT2018, had an IQR of approximately 10% (recall Figure 2.1b). Similarly to Section 3.4.2, the shape of the distribution of object scale change was assessed by comparing the area of the ground-truth bounding boxes for objects in consecutive frames of the VOT2016 dataset. However, unlike horizontal and vertical translation, it was found that object scale change was distributed similarly to a Gaussian distribution. Therefore, we sample from a Gaussian distribution with the aforementioned scale parameters.

Measure	Scale parameter (σ_s)				
	0.01	0.02	0.03	0.04	0.05
EAO	0.201	0.208	0.202	0.197	0.187
Average overlap	0.379 ± 0.105	0.381 ± 0.104	0.378 ± 0.103	0.368 ± 0.101	0.361 ± 0.100
Failures	1.483 ± 1.519	1.454 ± 1.480	1.474 ± 1.523	1.490 ± 1.524	1.510 ± 1.498

Table 3.6: Tracking performance for different scale parameters (σ_s) of the distribution from which object scale changes are sampled (π_s). The first, second, and third best performances shown in their respective colours.

Measure	Rotation parameter (σ_r)				
	$2\pi/64$	$3\pi/64$	$4\pi/64$	$5\pi/64$	$6\pi/64$
EAO	0.204	0.206	0.209	0.205	0.203
Average overlap	0.377 ± 0.106	0.379 ± 0.105	0.386 ± 0.109	0.383 ± 0.104	0.382 ± 0.099
Failures	1.433 ± 1.515	1.446 ± 1.489	1.463 ± 1.494	1.504 ± 1.549	1.509 ± 1.533

Table 3.7: Tracking performance for different scale parameters (σ_r) of the distribution from which object rotations are sampled (π_r). The first, second, and third best performances shown in their respective colours.

Table 3.6 shows the results of this experiment. Using a value of $\sigma_s = 0.02$ gives the best performance in terms of all three metrics. However, both $\sigma_s = 0.01$ and $\sigma_s = 0.03$ give roughly the same performance, with the EAO decreasing as the scale parameter increases. This is to be expected as the majority of the time an object changes very little in scale. Additionally, when taking into account local optimisation as well as scale change, it is likely that, particularly for smaller objects, the local optimisation takes care of any larger scale changes than can be sampled from the distribution. Given that $\sigma_s = 0.02$ gives the highest EAO value and lies within a range of high performing values, we select this to use in the final tracker.

We next evaluate PBTS-B using values of $\sigma_r \in \{ 2\pi/64, 3\pi/64, \ldots, 6\pi/64 \}$, corresponding to rotations of approximately 5°–17°. Unlike the previous experiment, we cannot reliably estimate the distribution of object rotations in the VOT2016 dataset. The bounding boxes in the dataset were fitted to the segmented object in each frame of video (Vojíř and Matas, 2017), meaning that if an object changed slightly between consecutive frames (e.g. due to deformation), it may have resulted in a bounding box that is at a completely different angle even though the object itself may have only rotated only a few degrees. We therefore assume that object rotations are distributed similarly to a Gaussian distribution.

Table 3.7 shows the results of this experiment. All values of σ_r evaluated have roughly the same level of performance in terms of EAO, with robustness decreasing

Measure	Local optimisation window size (W)				
	1	3	5	7	9
EAO	0.152	0.189	0.208	0.195	0.186
Average overlap	0.403 ± 0.101	0.351 ± 0.096	0.382 ± 0.103	0.354 ± 0.115	0.325 ± 0.117
Robustness	2.671 ± 2.764	1.602 ± 1.629	1.499 ± 1.556	1.389 ± 1.514	1.287 ± 1.449

Table 3.8: Tracking performance using different local optimisation window sizes. Note that $W = 1$ corresponds to not performing any local optimisation. The first, second, and third best performances shown in their respective colours.

slowly as the size of σ_r is increased. We suspect that this is because larger rotations, particularly for objects with a large aspect ratio, allows for parts that were previously matching the lower (or upper) region of the object to be rotated and matched against a similar background, while still keeping the central patches matching the object. Consequently this can lead to patch drift and eventually failure. We note that the different adjacent evaluated rotations, e.g. $3\pi/64$ and $4\pi/64$, will result sampling patch locations that are similar enough (in terms of patch positions) to have the local optimisation render them practically identical. While this mainly applies to smaller objects, as the distance between patches placed on it will be small, the majority of tracked objects are relatively small (Section 2.2.4). This may explain the relative insensitivity of all three metrics to σ_r. Therefore, we select $\sigma_r = 4\pi/64 = \pi/16$ to use in the final tracker as it lies within the middle of a range of high tracker performance.

Lastly, we evaluate PBTS-B using local optimisation window sizes of $W = \{1, 3, 5, 7, 9\}$, corresponding to allowing a patch to move up to $\{0, 1, 2, 3, 4\}$ pixels in any direction. A window size of $W = 1$ corresponds to only using the affine transform of the patches and not performing any local optimisation. The results of this experiment can be seen in Table 3.8. Performing no local optimisation ($W = 1$) severely limits the tracker's ability to model deformation, and, as demonstrated by the low EAO, leads to much worse tracking performance. Allowing patches to move (relatively) large amounts during local optimisation (i.e. $W = 7$ or $W = 9$) is also detrimental to performance. We suspect that this is because it allows too much freedom of movement and leads to patches attaching to distracting background regions and causing patch drift. This is reflected in the lower overlap and increased robustness and indicates an increase in predicted bounding box size. Predicting a larger bounding box naturally results a lower failure rate because it takes longer for

an object to move away from a poorly performing tracker with large bounding box than a smaller one. We would expect the robustness to continue to increase and the average overlap to continue decrease if the optimisation window (W) was to be further expanded. Increasing the freedom of patch movement further would allow patches to drift even faster, resulting in larger predicted bounding boxes, and so resulting in fewer tracking failures. Using a window size of $W = 5$ appears to give a good trade-off between the model being too rigid or too flexible and, therefore, we select this to use in the final tracker.

3.4.8 Summary

In this section we have proposed a two-part process for searching for the modelled object in subsequent frames. This generates non-shearing affine transformations of the patch locations in the previous frame and optimises their location within a small window. The motion of objects in the VOT2016 dataset is investigated, along with how predictive an object's motion is in a previous frame compared to the current one. It was found that an object's motion, in both the horizontal and vertical direction, was distributed in the form of a Laplace distribution. An object's motion between consecutive frames was found to be fairly well correlated, however there were many cases in which negative correlations occurred, indicating that the object moved in the opposite direction to the previous frame. Therefore, we investigated the use of a simple motion model to move the centre of the distribution used to generate candidate sets of locations from to be nearer the predicted location of the object. It was found that moving the centre of this distribution away from the location of the object was detrimental to the tracking performance.

Different matching schemes of local optimisation were also investigated. Our patch model was compared to using Normalised Cross Correlation with different template models. These were compared both in terms of overall tracking, and robustness in the form of analysing the IOU after tracking for very short sequences of frames. Our model performed best over entire videos and marginally worse than using NCC with RGB templates over short video sequences. The number of sets of

candidate locations to optimise L was also evaluated. Including local optimisation, i.e. $L > 0$, was found to be greatly beneficial to the tracking process and that only optimising the best $L = 100$ sets of candidate patches gave comparable performance to optimising all sets ($L = G$). We compared the Laplace distribution to the Gaussian for use in describing the horizontal and vertical movement of objects, as well as evaluating the use of Latin Hypercube Sampling to equally (proportional to the distribution being sampled) sample across parameter space. Both distributions were found to give approximately the same performance. However, the Gaussian distribution required a much larger scale parameter in order to adequately model the tails of the object's movement distribution. LHS provided a very small increase in performance, but should provide more stable sampling as it is far more likely to adequately sample the large movements that sometimes occur when tracking.

Lastly, we evaluated the object size (σ_s) and rotation (σ_r) scale parameters of the probability distributions, π_s and π_r respectively, from which candidate patch locations are generated, and also evaluated different sizes (W) of the local optimisation window. It was found that tracking performance is largely insensitive to both scale parameters. We suspect that this is because the local optimisation carried out after the affine transform has been applied allows for similar scale parameters to perform equivalently. For smaller objects in particular, the local optimisation effectively negates the differences between similar parameter values as the majority of transformations sampled will be close enough (in terms of pixel distance) to be made identical via the local optimisation. Different values of the window size (W) itself led to different performance of the tracker, with no local optimisation being carried ($W = 1$) out leading to much worse performance. This provides motivates the need for a part-based model, because without local optimisation the model is effectively a rigid holistic model only using subregions of the object to predict its match quality. Similarly, window sizes that were larger, and therefore allowing for more patches to move around more freely, also provided worse performance. This indicated that there was a happy medium between the model being overly rigid and allowing for too much non-affine motion.

3.5 Conclusion

In this chapter, we have proposed a novel method for part-based object tracking, and have detailed and evaluated two of its components, the object part model and object localisation scheme. Object parts are represented by a pairs of feature samples and their respective counts and empirically describe the feature distribution of the patches they model. We have compared several colour feature representations as well as comparing the model to a template update scheme. The number of patches, their size, and the number of samples used to represent them was also investigated, as well as different weightings for our modified Bhattacharyya distance measure. Given that a patch is represented by a set of samples of features extracted from the patches, there is plenty of scope for the use of different feature types, such as texture-based in the form of HOG (Dalal and Triggs, 2005), LBP (Ojala, Pietikäinen and Mäenpää, 2002), or SIFT (Lowe, 1999).

Object localisation in subsequence frames is carried out by a two-part process of first generating sets candidate locations by using non-shearing affine transforms and then optimising the location of the patches in the best matching sets of candidate locations. Object motion in the VOT2016 data was analysed, along with how predictive an objects motion in one frame is to describing its motion in a subsequent frame. In most videos objects move relatively small amounts between consecutive frames, however, more frequently than can be modelled with a Gaussian distribution, occasional large jumps occur. The local optimisation scheme was compared with a Normalised Cross Correlation template-based method, using grey-scale and RGB features, and its robustness over short sequences of video frames was analysed. Given that these experiments were carried out without using any form of update scheme, they show that the basic object part model works well. The distributions from which to sample the affine transform parameters from was also investigated, comparing the use of Gaussian and Laplace distributions, with and without Latin Hypercube Sampling (LHS). It was found that both distributions were capable of the same level performance, although the Gaussian distribution required much a larger scale parameter in order to adequately model the tail's of the object's movement

Component	Value
Patch features	RGB
Match radius (R)	20
Number of patches (P)	35
Patch side length (ω_x, ω_y)	5
Maximum number of samples (S_{max})	10
Modified Bhattacharyya Distance (b)	1.4
Sets of candidate patches to generate (G)	1000
Sets of candidate patches to locally optimise (L)	100
Local optimisation window size (W)	5
Sampling distribution (translation)	Laplace
Sampling distribution (size and rotation)	Gaussian
Horizontal scale parameter (σ_x)	0.15
Vertical scale parameter (σ_y)	0.1
Object size scale parameter (σ_s)	0.02
Rotation scale parameter (σ_r)	$\pi/16$

Table 3.9: The optimal tracking components and their attributes selected in this chapter.

distribution. LHS gave negligible increases in tracking performance, but should provide more stable tracking as it is far more likely to sample the larger movements that objects can sometimes make. For completeness, we summarise the optimal parameters chosen from the experiments in Table 3.9.

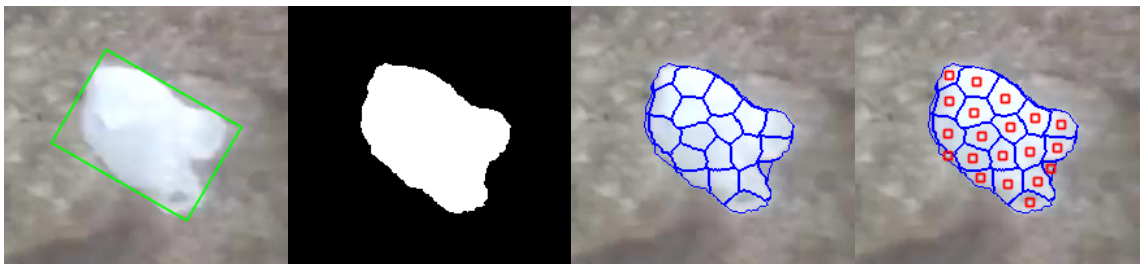
In the following chapter, the other components of the tracker will be described and evaluated. The patch model placement scheme, based on the first frame of a video and a bounding box denoting approximately where the object is located, will be detailed. The method will be compared to placing patches uniformly across the object's bounding box, as well as other placement strategies. The object's patch model initialisation and update schemes will also be described, along with experiments into their optimal parameter values. Lastly, an investigation will be carried out into methods for identifying drifting patches.

4. Part Placement, Initialisation, Update, and Drift

In the previous chapter we described the two main components, for a single-target, part-based tracker for tracking unknown objects in challenging sequences. These were the object's part model and localisation scheme, and were extensively evaluated and their components and parameters optimised. In this chapter, we describe enhancements to the basic tracking algorithm that are required in order to tailor the tracking scheme for the challenging, model-free tracking problems encountered in the real world. Specifically, in Section 4.1, we describe methods for placing object parts onto the object, rather than its background, for an image and a given bounding box. Given a selected object part location, how the part models are initialised is described in Section 4.2. Section 4.3 describes how they are updated once their location as been established in a future frame. Lastly, several methods to combat patch drift are investigated in Section 4.4. The chapter follows a similar structure to the previous one: each component is described and then experiments are carried out to evaluate its performance with respect to its different parameters and attributes. The testing protocol used is identical to that of the previous chapter and is outlined in Section 3.2.

4.1 Object Part Placement

In model-free tracking, the tracking algorithm is given the first frame of a video sequence and a region, typically a bounding box, that contains the object to be tracked. Unlike holistic, region-based trackers, that represent the object using the



(a) (b) (c) (d)

Figure 4.1: The patch location selection process. Given an image and a bounding box (green) indicating where the object is located (a), we segment the object using the alpha matting process described in Section 5.1.3. We then superpixel (blue) the image, discarding superpixels outside the masked region (c) and place patches (red) at the centre of the superpixels (d), given that they overlap less than a certain proportion of their area.

entire bounding box, part-based trackers have the additional problem of selecting where in the bounding box to place their parts. Therefore, in Section 4.1.1, we describe an object part location selection process that places patches in homogeneous regions of colour at the centre of superpixels within the given bounding box. We evaluate this method, optimise its parameters and compare it to an alternative placement scheme in Section 4.1.2.

4.1.1 Part Placement Scheme

As discussed in Section 2.1.4, one way to select locations of the object's parts is to choose regions of the bounding box that have desirable properties with respect to the part model. Given that we create compact models of patches based on their colour features, a desirable property in this case is to select homogeneous regions of colour. A common way to determine which image regions are homogeneous is to perform superpixeling (Ren and Malik, 2003). These techniques over-segment images into perceptually meaningful regions that are generally uniform in colour and texture. As superpixels tend to adhere to colour and shape boundaries, they also retain the image's structure (Stutz, Hermans and Leibe, 2018). Therefore, given an image and a bounding box indicating the object's location, we superpixel the image and select object part locations that are at the centre of superpixels that reside within the object's predicted location (as determined by the alpha matting segmentation scheme described in Section 5.1.3). The object's part models are then initialised

Algorithm 4 Patch location selection

Input: I : Image
$\mathcal{F}$: Predicted object mask
ω_x : Patch width
ω_y : Patch height
P : Number of patches
γ : Patch overlap threshold
Output: $\mathcal{L}$: Patch centroid locations

1: Calculate number of superpixels needed to ensure the object's mask has approximately P superpixels when I is segmented. *(Equation 4.1)*
2: Apply the SLICO superpixeling algorithm to I; this returns a label mask denoting which pixels belong to which superpixel.
3: Set the label of all superpixels that lie completely outside $\mathcal{F}$ to have the same label (0).
4: Calculate the area of each superpixel
5: $\mathcal{L} \leftarrow \varnothing$
6: **while** There are still remaining superpixels **and** $|\mathcal{L}| < P$ **do**
7: Select the superpixel with the largest area
8: Calculate its centroid $\mathbf{c}_p$
9: Set Ω_p to be a rectangular region of height ω_y and width ω_x, centred on $\mathbf{c}_p$
10: **if** Ω_p has an overlap proportion $< \gamma$ with all other placed patches **then**
11: $\mathcal{L} \leftarrow \mathcal{L} \cup \{ \mathbf{c}_p \}$
12: **end if**
13: **end while**

(Section 4.2) at the selected locations, and the object is tracked in subsequent frames. An overview of the patch placement algorithm is shown in Figure 4.1.

Given the object's predicted location mask $\mathcal{F}$ (Figure 4.1b) and the desired number P of patches of width ω_x and height ω_y, we generate a set $\mathcal{L}$ of P patch locations using Algorithm 4. It first calculates the number N_{sp} of superpixels that need to be produced by the superpixeling algorithm in order to ensure that roughly P superpixels lie within the object's region $\mathcal{F}$

$$N_{sp} = \left\lfloor \frac{P \cdot |\mathcal{F}|}{U} \right\rceil, \tag{4.1}$$

where $\lfloor \cdot \rceil$ performs integer rounding and U is the number of non-zero elements (i.e. pixels labelled as belonging to the object) in $\mathcal{F}$.

A parameterless version (SLICO[1]) of the SLIC superpixeling algorithm (Achanta et al., 2012) is then applied to the image I. This generates a mask

[1] `https://ivrl.epfl.ch/research-2/research-current/research-superpixels/#SLICO`

containing a label for each pixel that indicates which pixel belongs to which superpixel. The SLIC algorithm was selected as it is computationally efficient and performs well in segmentation benchmarks when compared to recent state-of-the-art methods (Stutz, Hermans and Leibe, 2018). The zero-parameter version of SLIC allows for each superpixel to have its own compactness parameter, which means that regular shaped superpixels are generated for both smooth and rugged image regions. This is an important property, because the optimal compactness parameter (i.e. how smooth or rugged the superpixels should be in order to fit the image regions well) will not be the same for every image, and may not even be the same for each superpixel in a single image.

The labels of superpixels lying completely outside $\mathcal{F}$ are set to have the same background label, leaving only those inside and on the border of $\mathcal{F}$ as candidates for patch placement (Figure 4.1c). We then place a patch at the centroid of the largest available superpixel if doing so would cause the patch to overlap with all other patches with a proportion of its area less than γ (Algorithm 4 lines 7-12). This process is then repeated until either P patch locations have been selected or there are no more superpixels left to consider.

The amount overlap γ allowed to occur between patches, relative to their area, controls the density of the patches across the object. For larger objects, where patches do not reach the boundaries of superpixels, this value has no effect. However, for smaller objects, patches may overlap multiple superpixels. This can occur when the area of the superpixels are less than that of the patches' area and when the superpixels are roughly rectangular with dimensions shorter or longer than the patches. Limiting the patch overlap in these cases reduces the amount of repeated information in neighbouring patches. We investigate the effect of γ in tracking, as well as an alternative placement scheme, in the following section.

4.1.2 Alternative Placement Method

In order to validate our assertion that tracking homogeneous regions of colour is desirable, we compare it to tracking inhomogeneous regions. Superpixels are

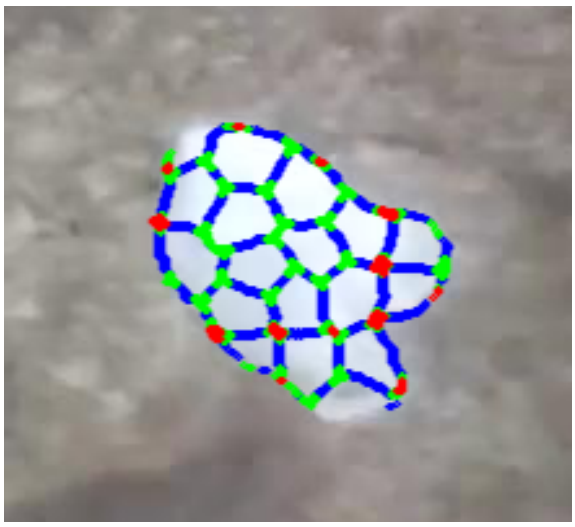
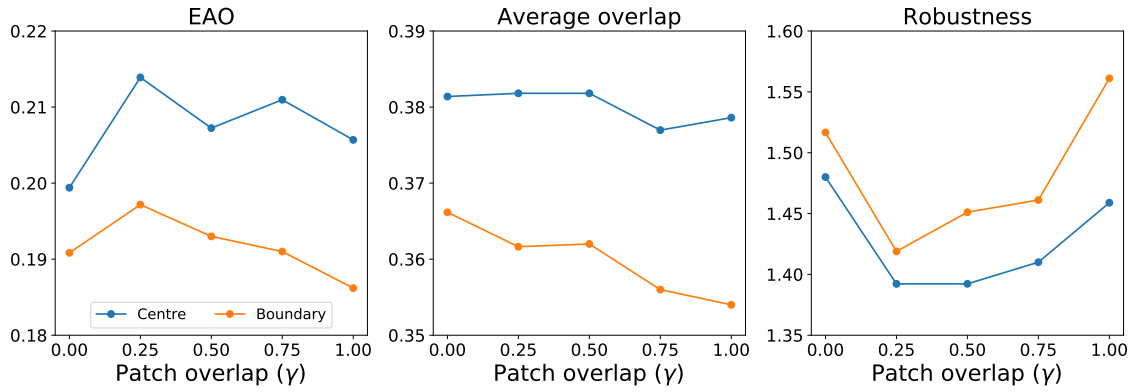


Figure 4.2: Superpixel boundaries, coloured by how many superpixels a patch of size 7×7 pixels overlaps. A colour of blue, red, and green correspond to overlapping 2, 3, or 4 superpixels respectively.

generally homogeneous in colour and, therefore, their boundaries are ideal locations to centre inhomogeneous patches because this results in them intersecting multiple, homogeneous regions. An example of the number of superpixels a patch can overlap if placed on the boundary between superpixels can be seen in Figure 4.2. Therefore, we create a superpixel edge placement method similar to Algorithm 4, but we modify lines 7-8 of the algorithm to select the next potential superpixel location $\mathbf{c}$ to be on the boundary between superpixels that allows for a $\omega_x \times \omega_y$ sized patch to overlap the most superpixels. In order to investigate the effect of allowing different amounts of overlap γ between patches we also vary γ. These experiments are carried out using PBTS-B (Table 3.1).

Figure 4.3 shows the performance of the two patch placement strategies for different levels of allowable patch overlap. As can be seen from Figure 4.3a, tracking patches that were initially placed at the centre of superpixels is superior to tracking patches placed to overlap as many superpixels as possible. It might be expected that tracking regions that contain a more varied colour representation (as would be the case of overlapping superpixels) would result in better tracking performance as it is less likely that there are any other regions in the image that contain that exact mix of colour. We suggest that the reason that this is not the case may be due to a combination of the low image quality of the VOT2016 dataset and the general



(a) (b) (c)

Figure 4.3: Tracking performance for the two patch placement strategies with varying levels of allowable patch overlap (γ). The EAO (a), average overlap (b), and robustness (c) is shown for placing patches at the centre of superpixels (blue) and placing patches to overlap as many superpixels as possible (orange).

difficulty of the tracking videos. If part of a tracked, homogeneous, region changes, e.g. due to motion blur or illumination, the rest of the region still matches the model. However, if a part of a tracked inhomogeneous region changes then the model will only be able to match the unchanged region, resulting in a lower match quality.

Allowing partial overlap of patches also appears to have performance benefits, with average overlap (Figure 4.3b) being fairly consistent across lower proportions of patch overlap, as well as an increase in robustness (Figure 4.3c). This latter increase may be due to having a larger number of patches allowed in small objects, because, if an object is sufficiently small, it would not be possible to fit an arbitrary number of patches on it. It is also interesting to note the decrease in robustness when $\gamma \geq 0.75$. We suggest this may due to patch locations being selected that are nearer the centre of the object because those are likely to be larger (e.g. Figure 4.2). This would result in a less even spread of patches over the object, leading to diminished robustness. Therefore, we choose to place patches at the centre of superpixels (Algorithm 4), with an allowable overlap between patches during initialisation of $\gamma = 0.25$.

4.2 Patch Model Initialisation

In this section we describe a sampling scheme to select features to include in a patch's model (Section 4.2.1). Section 4.2.2 compares this with two other sampling schemes.

Algorithm 5 Patch model initialisation

Input: $\Omega_p \in I$: Patch region
R : Match radius
S_{max} : Maximum number of samples
Output: $\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_{S_{max}}, C_{S_{max}}) \}$: Patch model

1: $I_i \leftarrow \text{choose}(\Omega_p)$ *Randomly select a pixel I_i from Ω_p, with features $\mathbf{x}_i$*
2: $\mathcal{M}_p \leftarrow \{ (\mathbf{x}_i, 1) \}$
3: $\Omega_p \leftarrow \Omega_p \setminus I_i$
4: **while** $\Omega_p \neq \varnothing$ **do**
5: $I_i \leftarrow \text{choose}(\Omega_p)$ *Randomly select a pixel I_i from Ω_p*
6: $\mathcal{D} = \{ \|\mathbf{x}_i - \mathbf{f}_j\| < R \mid \mathbf{f}_j \in \mathcal{M}_p \}$
7: **if** $\mathcal{D} = \varnothing$ **then** *If it does not match any feature in the model*
8: $\mathcal{M}_p \leftarrow \mathcal{M}_p \cup \{ (\mathbf{x}_i, 1) \}$ *Add it to the model*
9: **else** *Find the closest feature to it*
10: $(\mathbf{f}_j, C_j) = \underset{(\mathbf{f}_j, C_j) \in \mathcal{M}_p}{\text{argmin}} \{ \|\mathbf{x}_i - \mathbf{f}_j\| \mid \mathbf{f}_j \in \mathcal{M}_p \}$
11: $(\mathbf{f}_j, C_j) \leftarrow (\mathbf{f}_j, C_j + 1)$ *Increment the closest matching feature's count*
12: **end if**
13: $\Omega_p \leftarrow \Omega_p \setminus I_i$
14: **end while**
15: **while** $|\mathcal{M}_p| > S_{max}$ **do**
16: Remove feature-count pair from $\mathcal{M}_p$ that has the lowest count
17: **end while**

4.2.1 Model Initialisation Scheme

Once the patch locations have been selected (Section 4.1), a model for each patch is created. A patch is characterised by a model $\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_S, C_S) \}$, containing S pairs of features $\mathbf{f}_s$ and their corresponding match counts C_s. The match counts indicate how many pixels within the patch have features that match $\mathbf{f}_s$, where a pixel I_i with features $\mathbf{x}_i$ is said to match the sample if $\|\mathbf{x}_i - \mathbf{f}_s\| < R$. A full description of the patch model can be found in Section 3.3.1.

Given a patch region Ω_p and maximum number S_{max} of samples allowed in the model, we model a patch according to Algorithm 5. It starts by randomly selecting a pixel I_i and uses its features $\mathbf{x}_i$ as the first sample in the model $\mathbf{f}_1 = \mathbf{x}_i$, with match count $C_1 = 1$ (Algorithm 5, lines 1-3). Another pixel is randomly selected and its features $\mathbf{x}_i$ are compared to those in the model to see if it matches (Algorithm 5, line 6). If it does not match any features in the model, i.e. is greater than R away from all of them, then it is added to the model with an associated count of one. If it does match a feature $\mathbf{f}_j$, then its count C_j is incremented, and if it matches

multiple features then the one it is closest to has its count incremented. This process is repeated until all pixels in the patch have been evaluated once. Finally, if there are more than S_{max} feature-count pairs in the model, then the pairs with the lowest counts are removed until the model only contains S_{max} pairs of features and counts.

This method performs a kind of pseudo-clustering on the data without the need for an explicit clustering algorithm tailored to producing clusters of a fixed width ($2R$), centred on pixel features. Consequentially, this method is much less computationally intensive than traditional clustering techniques. Given that the regions that are initially modelled by the patches are relatively homogeneous, and therefore close together in colour space, the stochastic sampling is expected to create similar sets of samples for each run on a given patch. Limiting the number of samples in each model effectively reduces over-fitting in the model by excluding rare samples that do not characterise the patches. The random sampling scheme is compared to two, more principled, ways of selecting features for the model in the following section.

4.2.2 Feature Sample Selection Methods

Here we compare the random selection scheme to the two most extreme configurations of where the selected features lie in feature space, i.e. selecting features such that they occupy the largest or smallest amount of feature space possible. This is carried out in order to ascertain whether there are any performance benefits, or losses, to preferring a certain configuration of features over any other. We carry this out using PBTS-B (Table 3.1).

The first method selects features in such a way as to create a model that is as compact as possible. It starts by selecting the first feature sample $\mathbf{f}_1$ in the model as those from the pixel, with the feature vector $\mathbf{x}_i$, that is closest to the mean of all the pixel features in the patch. It then selects the next feature sample to add as being the pixel's features that are closest to, but not matching, any other features already in the model. This process is repeated until either all the features of pixels in the patch have been added to the model, or they all match a feature in the model.

Method	EAO	Average overlap	Robustness
Random	0.211	0.384 ± 0.108	1.423 ± 1.479
Compact	0.209	0.382 ± 0.101	1.437 ± 1.508
Expansive	0.201	0.379 ± 0.104	1.474 ± 1.557

Table 4.1: Tracking performance of the three different model feature selection schemes. The methods are: random selection of samples (Random), and selecting samples to occupy the least (Compact) and most (Expansive) amount of feature possible. The first, second, and third best performances shown in their respective colours.

Finally, the features of each pixel in the patch are compared to those in the model and the closest feature's count corresponds to how many pixel features are the closest to it. This covers the least amount of feature space possible and is labelled as the *compact* representation.

A very similar sample selection scheme is carried out for the second method. However, it selects features with the goal of covering as much feature space as possible. This method selects the initial feature sample to be included in the model as the one closest to the origin, as this is guaranteed to be on the convex hull of the set of pixel's features. It then repeatedly selects the next feature sample to be included in the model as the pixel's features that are the farthest away from all features in the model. Once the features have been selected, it continues identically to the *compact* method, and counts the number of features in the match that match each feature. We label this the *expansive* representation.

As can be seen from Table 4.1, all three initialisation strategies perform similarly with respect to all three performance measures. This is an interesting result because one might expect that either having a compact model, i.e. matching a smaller range of values, or having an expansive model, i.e. matching a larger range of values, would be preferable. We suggest that this is due to the homogeneous nature of the patches modelled, meaning that the specific samples used to represent the patch is somewhat unimportant. Given the additional distance computations needed for both the compact and expansive methods, we choose to use the random sampling method for initialisation. Similarly to the previous chapter, we note that the size of the standard deviations of the tracker's robustness is large but that their relative sizes between different parameters is still a useful for comparison.

4.3 Patch Model Update

After the optimal patch positions in the current frame have been identified by the object localisation scheme (Algorithms 2 and 3, Section 3.4), each patch's model needs to be updated to incorporate any new information about the object's appearance. Therefore, in this section, we describe an update scheme to the patch models (Section 4.3.1) which updates both the feature sample locations and counts. Optimal update rates for both updated components of the model are determined in Section 4.3.2.

4.3.1 Model Update Scheme

Tracking methods that represent objects with histograms, such as FragTrack (Adam, Rivlin and Shimshoni, 2006), ANT (Čehovin, Leonardis and Kristan, 2016a), or BHMC (Kwon and Lee, 2009), update their models (histograms) by linearly interpolating its bin counts between their current model and the newly matched image region (see Section 2.1.2 for further details). Given that our our model also contains counts of the number of matches, it can be updated in a comparable manner. Histogram-based methods can also store any information about newly seen features in a patch because they contain bins for all possible locations of feature space. Our patch model, however, is a sparse representation of the patch features and therefore does not already have the capability to store information about new features seen in the patch. Consequently, we present a new update scheme that can introduce (and remove) feature-count pairs from the model, as well as update counts and features.

Given a patch's new location $\mathbf{c}_p$ and its model $\mathcal{M}_p$, we update both its features $\mathbf{f}_j$ and counts C_j using Algorithm 6. It first updates $\mathcal{M}_p$ based on the pixels Ω_p^+ within the patch that match the model, and then creates a new patch model (Algorithm 5) based on the pixels Ω_p^- that do not match any features in the model and incorporates it into the current model.

The algorithm starts by separating the patch Ω_p into two sets of pixels, those that match at least one sample in the model, Ω_p^+, and those that do not match any samples Ω_p^-. Then, similarly to evaluating patch quality (Algorithm 1), it creates

Algorithm 6 Update patch model

Input: I : Image
ω_x : Patch width
ω_y : Patch height
$\mathbf{c}_p$: Patch location
$\mathcal{M}_p = \{ (\mathbf{f}_1, C_1), ..., (\mathbf{f}_S, C_S) \}$: Patch model
R : Match radius
β_c : Counts update rate
β_s : Samples update rate
Output: $\mathcal{M}_p$: Updated patch model

1: Set Ω_p to be a rectangular region of height ω_y and width ω_x, centred on $\mathbf{c}_p$
2: $\Omega_p^+ \leftarrow \{ I_i \mid \|\mathbf{x}_i - \mathbf{f}_j\| < R \forall I_i \in \Omega_p \}$ *Matches a sample in* $\mathcal{M}_p$
3: $\Omega_p^- \leftarrow \Omega_p \setminus \Omega_p^+$
4: $\mathbf{p} \leftarrow [0, \ldots, 0]$ $|\mathbf{p}| = |\mathcal{M}_p| = S$ *and* p_j *contains*
5: **for** $I_i \in \Omega_p^+$ **do** *the number of matches to* $\mathbf{f}_j$
6: $\mathbf{f}_j = \underset{\mathbf{f}_j \in \mathcal{M}_p}{\operatorname{argmin}} \{ \|\mathbf{x}_i - \mathbf{f}_j\| \mid \mathbf{f}_j \in \mathcal{M}_p \}$
7: $p_j \leftarrow p_j + 1$
8: **end for**
9: **for** $(\mathbf{f}_j, C_j) \in \mathcal{M}_p$ **do**
10: $C_j \leftarrow (1 - \beta_c) C_j + \beta_c p_j$ *Linearly interpolate between old and new counts*
11: **if** $p_j > 0$ **then**
12: $\mathbf{f}_j \leftarrow$ Mean feature vector of pixels matching $\mathbf{f}_j$
13: $\mathbf{f}_j \leftarrow (1 - \beta_s)\mathbf{f}_j + \beta_s \tilde{\mathbf{f}}_j$ *Update sample*
14: **end if**
15: **end for**
16: Remove all feature-count pairs with $C_j < \beta_c$ from $\mathcal{M}_p$
17: Create model $\mathcal{M}_p^-$ from Ω_p^- *(Algorithm 5)*
18: **for** $(\mathbf{f}_j^-, C_j^-) \in \mathcal{M}_p^-$ **do** *Combine the patch models*
19: $C_j^- \leftarrow \beta_c C_j^-$
20: $\mathcal{M}_p \leftarrow \mathcal{M}_p \cup \{ (\mathbf{f}_j^-, C_j^-) \}$
21: **end for**

a vector $\mathbf{p}$ whose j-th element p_j corresponds to the number of matches each pixel feature $\mathbf{x}_i$ has to the model feature $\mathbf{f}_j$ (Algorithm 5, lines 4-8). Note that in the event of a pixel $I_i \in \Omega_p^+$ matching multiple features, only the closest match is counted. Each count C_j in the model is then linearly interpolated towards the corresponding newly observed count p_j using an update rate of β_c (Algorithm 5, line 10). If the corresponding feature $\mathbf{f}_j$ in the model has pixels that match it (i.e. $p_j > 0$), we update its position. This is carried out by linearly interpolating between $\mathbf{f}_j$ and the mean feature vector of the features of pixels who match $\mathbf{f}_j$, using an update rate of β_s (Algorithm 5, lines 11-14).

The use of two separate update rates, β_c and β_s allows for the features samples

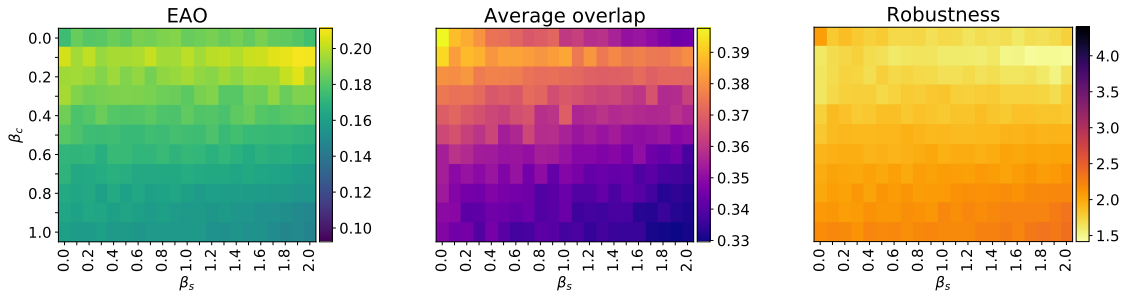
and counts to be adapted at different speeds. This is an important property as when an image region changes colour due to illumination the brightness of each colour in the region increases, rather than the relative proportion of colours. Updating $\mathbf{f}_j$ has the effect of moving the centre of the sphere containing matching samples towards (or past if $\beta_s > 1$) the matches' centre of mass. It allows the model to *follow* the region of colour space that it characterises as it changes over time. In the standard histogram-based approach, when the features change over time they move from matching one bin to another. When this occurs there will be a sudden loss of probability mass within the histogram, resulting in poorer matching as a result.

New information about the patch, i.e. the pixels Ω_p^- whose features do not match any features in the model, are then incorporated into the model (Algorithm 5, lines 17-21). A new patch model $\mathcal{M}_p^-$ is created for the pixels in Ω_p^-, and each of its counts C_j^- are scaled by β_c. Its feature-count pairs are then added to the model. The counts are scaled in order to match the behaviour of count updating (Algorithm 5, line 10), as this introduces the feature-count pair as though they were already in the model with a previous count of 0.

Lastly, the feature-count pairs with $C_j < \beta_c$ are removed from the model. This allows for features that have not been seen for a while to be removed from the model in order to limit the model's size and computational complexity. A threshold of β_c was chosen as it is the smallest value that a new count can have if it has been added to the model in the same time-step, i.e. it had a value of $C_j = 1$ before scaling.

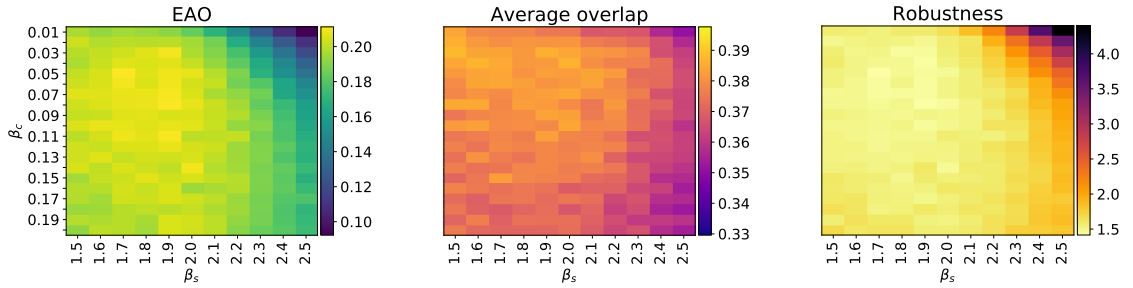
4.3.2 Model Update Rates

In this section, we investigate the optimal values of both the feature β_s and count β_c update rates, also known as learning rates. We evaluate PBTS-B (Table 3.1) with a large range of update rates for both parameters, $\beta_c = \{\, 0, 0.1, \ldots, 1 \,\}$ and $\beta_s = \{\, 0, 0.1, \ldots, 2 \,\}$. Sample update rates of $\beta_s > 1$ are included as these correspond to predicting where the tracker would expect the colour features to be in the subsequent frame. Because β_s controls the linear interpolation between the feature sample's current value $\mathbf{f}_j$ and the mean $\tilde{\mathbf{f}}_j$ of the pixels' features that match it, we



(a) (b) (c)

Figure 4.4: Tracking performance with respect to different update rates for both the features (β_s) and counts (β_c). The EAO (a), average overlap (b), and robustness (c) for different rates is shown, with a brighter (more yellow) colour reflecting better performance for the respective performance metric.



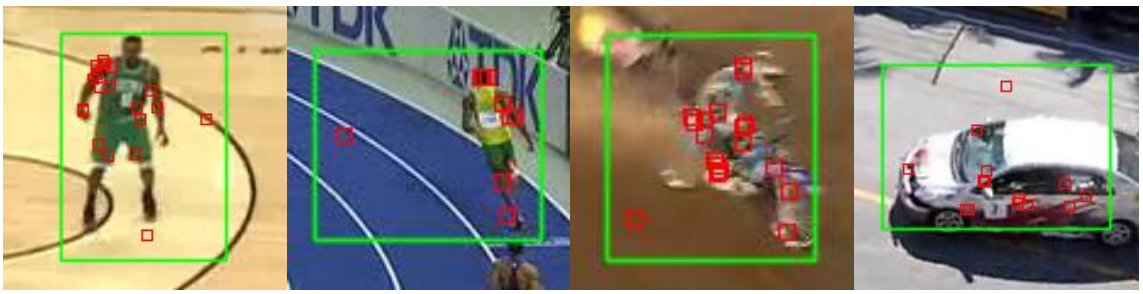
(a) (b) (c)

Figure 4.5: Tracking performance with respect to different update rates for both the features (β_s) and counts (β_c), using a smaller step size between parameter values and focusing on the region of high performance in Figure 4.4. The EAO (a), average overlap (b), and robustness (c) for different rates is shown, with a brighter (more yellow) colour reflecting better performance for the respective performance metric.

initially limit this rate to a maximum of $\beta_s = 2$. If there is a large change in colour between two consecutive frames, such that $\|\mathbf{f}_j(t) - \tilde{\mathbf{f}}_j(t+1)\| = R$ and in the next frame there is no change in colour, i.e. $\mathbf{f}_j(t+2) = \tilde{\mathbf{f}}_j(t+1)$, a value of $\beta_s > 2$ would result in $\tilde{\mathbf{f}}_j(t+1)$ no longer matching the model.

Figure 4.4 shows the performance for all combinations of β_c and β_s. Note that the region containing the highest values of EAO (Figure 4.4a), $\beta_c \in [0.1, 0.2]$ and $\beta_s \in [1.7, 2.0]$, resides on the edge of the parameter space evaluated. There is also a sharp decline in performance between $\beta_c = 0$ and $\beta_c = 0.1$, indicating that additional resolution is required in this region. Therefore, we additionally evaluate $\beta_c = \{\, 0, 0.01, \ldots, 0.2 \,\}$ and $\beta_s = \{\, 1.5, 1.6, \ldots, 2.5 \,\}$.

The results of the fine-grained parameter evaluation can be seen in Figure 4.5. It is interesting to note that performance degrades quite rapidly for $\beta_s > 2$ for all



(a) (b) (c) (d)

Figure 4.6: Examples of patch drift in the `basketball` (a), `bolt1` (b), `motocross1` (c), and `racing` (d) videos taken from the VOT2016 dataset. In each figure the tracked patches are shown in red and the tracker's predicted bounding box is shown in green. Note that the vast majority of patches are successfully tracking the object, but that the inclusion of only a few drifting patches substantially increases the predicted bounding box.

values of β_c. This confirms our interpretation of β_s, as the tracker starts to suffer far more failures (Figure 4.5c) as β_s increases beyond 2. As the results of Figure 4.5a show, it is beneficial to use a much lower of updating for the counts, and much larger rates for adapting the feature values themselves. These results suggest that the proportions of a particular feature in a patch, relative to others features, should change relatively little, whereas the feature values (the colours themselves) should be able to change more rapidly due to the challenging tracking conditions. For this reason, we choose to use update rates of $\beta_s = 1.7$ and $\beta_c = 0.05$ as these lie within a large region of parameter space of high performance.

4.4 Object Part Drift

Similar to other patch-based trackers, our tracker encounters situations in which one or more patches can drift away from the target. Tracking drift is reviewed in more detail in Section 2.1.3. Figure 4.6 shows several examples of this, where the majority of patches are correctly tracking the object but there are several that have drifted on to the background. Therefore, in this section, we investigate methods for identifying poorly performing (i.e. drifting) patches, based on two of a patch's main characteristics, its match quality and its location relative to other patches in the model. We also investigate a method for selecting new locations for patches that have been identified as drifting.

4.4.1 Match Quality Outliers

The quality (Equation 3.4) of a set of patch locations indicates how good a match they are to their respective patch models. We select the best of these to be the patch's location in the current frame (Section 3.4.1). This may, however, include several patches that have a poor match quality (i.e. a high value of MBD, Equation 3.2). Consequently, because we use the patch locations to predict the bounding box of the object, this may result in a poor estimation of the object's location even though the majority of patches are correctly tracking the object.

In order to counteract this effect, using PBTS-B (Table 3.1), we investigate discarding patches that are have sufficiently poor match quality from the bounding box calculation. We label these patches *outliers*. We determine whether a patch is an outlier based on thresholding its match quality (Equation 3.3) such that if $Q\left(\mathcal{M}_p, \tilde{\mathbf{c}}\right) < T$, for some threshold $T \in [0, 1)$, then the patch would be marked as an outlier and not used in the calculation of the bounding box of the object. Note, however, that the patch continues to be updated in the same way as other patches. We label this method as *match quality thresholding* (MTQ). As it is not obvious whether to use the outlier patches in predicting a set of patches' overall quality in the next frame of video, we also investigate excluding the contribution of patches, marked as outliers in the previous frame, in the current frame's match quality calculations. This second scheme is in some ways similar to the modified Bhattacharyya distance (Equation 3.2), however, this excludes the contribution of poorly performing patches, rather than down-weighting them. We denote this method MTQi. This latter method bears some resemblance to that of one of the patch outlier methods discussed in Adam, Rivlin and Shimshoni (2006). However, unlike them, when a patch has a match quality below a certain threshold, rather than setting its quality to equal the threshold, we discard the patch's match quality completely.

Figure 4.7 shows the performance of the tracker using MQT (blue) and MQTi (orange) for thresholds $T = \{\, 0.05, 0.1, \ldots, 0.5 \,\}$, compared with not using any outlier detection (dashed). These results show that using either MQT and MQTi appears to have little effect on the tracker using lower levels of the threshold T and is

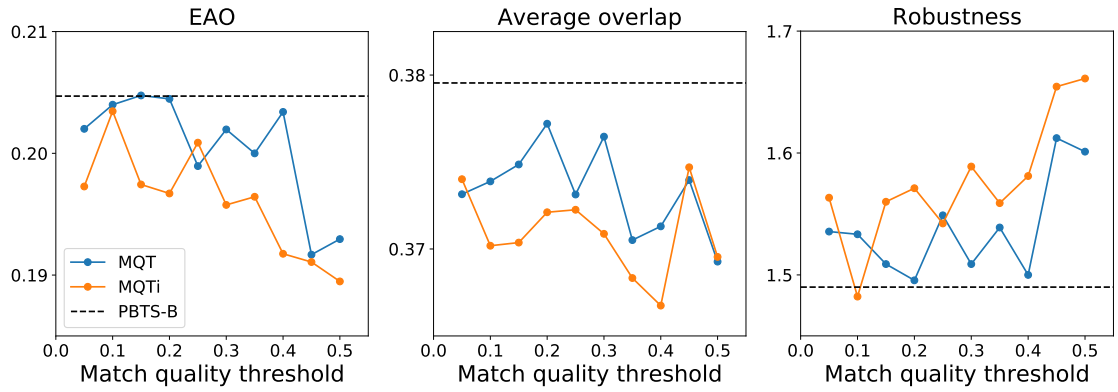


Figure 4.7: Tracking performance when predicting an object's location (bounding box) when discarding the location of patches with a match quality below a certain threshold (MQT). This is also compared with additionally ignoring a patch's match quality when calculating the overall quality of a set of patches in the next frame (MQTi). The dashed line represents the performance of the tracker without either modification.

slightly detrimental at higher thresholds. This suggests that a patch's match quality is not a good indicator of its likelihood of being an outlier. It is interesting to note the lack of performance difference MQT and MQTi, as one might expect that discarding the contribution of patches we know to be of poor quality in the previous frame when evaluating a new set of patches would have a positive effect on tracking performance. This suggests that most patches are approximately the same quality and therefore make the same contribution regardless of whether their quality is included in the calculation of the overall match quality. Given that no threshold T for either MTQ or MTQi improved tracking performance, we choose not to include likelihood thresholding in the final tracker. However, we select the best performing technique, MQT with $T = 0.15$, to evaluate with the addition of the patch replacement framework described in Section 4.4.3.

4.4.2 Patch Location Outliers

The distribution of the set of patches' horizontal and vertical positions can provide useful insight as to whether a patch is an outlier. One way to determine this is to estimate the scale of the distribution of locations, and label standardised locations that are further away from the distribution's location than some multiple of the scale as being outliers. Since we want to minimise the effect outliers have on estimating

the scale of a distribution, we estimate scale using *robust* estimators. These are said to be robust as they give the same measure of scale even if a certain proportion (known as their breakdown point) of the extreme values used in estimation are replaced by arbitrarily sized outliers (Rousseeuw and Leroy, 1987). For example, a robust location estimator is the median, which robustly estimates the location of a distribution. This is in contrast to the mean, which is skewed by the presence of any outliers. In this section, we explore the use of three robust scale estimators, MAD, S_n, and Q_n, for use in detecting drifting patches.

One of the most widely known robust scale estimators is the *median absolute deviation about the median* (Hampel, 1974), also known by its shorter name, *median absolute deviation* (MAD). If we have a set of samples taken from a distribution, in this case the horizontal components of the patch locations $\mathcal{X} = \{ x_1, x_2, \ldots, x_P \}$, then we denote its sample median $\text{med}(\mathcal{X})$ as the middle order statistic (i.e. the middle value of a sorted set containing all values of $\mathcal{X}$) when $P = |\mathcal{X}|$ is odd, and the arithmetic mean of the middle two values when P is even. The median absolute deviation is defined in terms of the sample median:

$$\text{MAD}(\mathcal{X}) = b_n \, \text{med}(\{ \, |x_i - \text{med}(\mathcal{X})| \, \forall x_i \in \mathcal{X} \, \}) \tag{4.2}$$

where b_n is a scale factor. We set $b_n = 1.4826$ as this puts MAD on the same scale (i.e. magnitude) as the standard deviation if the data were normally distributed (Rousseeuw and Croux, 1993). This can then be used for outlier detection by computing

$$\tilde{x}_i = \frac{|x_i - \text{med}(\mathcal{X})|}{\text{MAD}(\mathcal{X})} \tag{4.3}$$

for each $x_i \in \mathcal{X}$ and labelling x_i as an outlier if $\tilde{x}_i > K$. If $K = 1$, for example, then $\tilde{x}_i > 1$ corresponds to $\tilde{x}_i$ being more than one length scale away from the median of the data.

One of the main criticisms of using the MAD is that it is a symmetrical estimator about a location estimate (the median of the data) and therefore is less suitable for skewed distributions. Rousseeuw and Croux (1993) proposed two new

Correction factor	**Number of data points** (P)							
	2	3	4	5	6	7	8	9
c_n	0.743	1.851	0.954	1.351	0.993	1.198	1.005	1.131
d_n	0.399	0.994	0.512	0.844	0.611	0.857	0.699	0.872

Table 4.2: Scale correction factors for the S_n (c_n) and Q_n (d_n) estimators, for a given data set of size P. These factors scale the value of the estimator so that it is on the same scale (magnitude) as the standard deviation of the data (Croux and Rousseeuw, 1992).

scale estimators, S_n and Q_n, that are both suitable for asymmetric distributions and have the same breakdown point as the MAD (50%). The S_n robust scale estimator is defined as

$$S_n(\mathcal{X}) = c_n \, 1.1926 \, \underline{\text{med}}(\{ \, \overline{\text{med}}(\{ \, |x_i - x_j| \, \forall x_j \in \mathcal{X} \mid i \neq j \, \} \, \forall x_i \in \mathcal{X}) \, \}), \tag{4.4}$$

where $\underline{\text{med}}(\mathcal{X})$ and $\overline{\text{med}}(\mathcal{X})$ correspond to the lower and upper medians of $\mathcal{X}$ respectively when P is even and are equal to $\text{med}(\mathcal{X})$ when P is odd. The lower and upper median of a sequence of even length are its middle-left and middle-right values respectively. The value of c_n plays an identical role to that of b_n in Equation (4.2) and ensures that it is on the same scale as the standard deviation (Croux and Rousseeuw, 1992). Its value for $P \leq 9$ is defined in Table 4.2, and for $P > 9$ has values of:

$$c_n = \begin{cases} \frac{P}{P-0.9} & P \text{ odd} \\ 1 & P \text{ even.} \end{cases} \tag{4.5}$$

The final robust scale measure, Q_n, is defined as

$$Q_n(\mathcal{X}) = d_n \, 2.2219 \, \{ \, |x_i - x_j| \forall x_i, x_j \in \mathcal{X} \mid i < j \, \}_{(k)} \, , \tag{4.6}$$

where $k = \binom{h}{2}$, $h = \frac{P}{2} + 1$ and the notation $\{ \cdots \}_{(k)}$ denotes the k-th order statistic of the set of values. The correction factor d_n, in a similar fashion to c_n, takes the values of those in Table 4.2 when $P \leq 9$ and for $P > 9$ is defined (Croux and Rousseeuw, 1992) as

$$d_n = \begin{cases} \frac{P}{P+1.4} & P \text{ odd} \\ \frac{P}{P+3.8} & P \text{ even.} \end{cases} \tag{4.7}$$

Both robust estimators, S_n and Q_n are provably more efficient, i.e. require fewer observations to obtain the same accuracy of estimation, than the MAD estimator (Rousseeuw and Croux, 1993). This means that, in theory, they should provide a better estimation of whether a patch is an outlier for a fixed number of patches, based on the distribution of their locations. The main difference between the two measures is that Q_n is more efficient than S_n and can be computed faster, but S_n is less biased for smaller P. Note that this latter difference is only applicable if the distribution of the data is non-Gaussian, as the correction factors (c_n and d_n) correct any bias present for Gaussian data.

In the following investigation we compare the performance of the three aforementioned scale estimation methods to detect outliers such that patch p centred at $\mathbf{c}_p = [x_p, y_p]^T$ is marked as an outlier if

$$\frac{|x_p - \text{med}(\mathcal{X})|}{\text{EST}(\mathcal{X})} > K \qquad \text{or} \qquad \frac{|y_p - \text{med}(\mathcal{Y})|}{\text{EST}(\mathcal{Y})} > K, \tag{4.8}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are sets containing the horizontal and vertical locations of the patches respectively. K is a threshold denoting how many multiples of the scale estimator that the patch's location must be further away from the median than, and $\text{EST}(\mathcal{X})$ is one of the three scale estimators. We evaluate each estimator with a threshold of $K = \{\, 1, 2, 3, 4, 5 \,\}$ and use these as the basis for four methods of outlier detection:

1. Labelling a patch as an outlier if either threshold in Equation (4.8) is reached (labelled MAD, Qn, and Sn).

2. Same as above but also ignoring the outlier patches in the next frame to calculate the optimal set of candidate patches (labelled MADi, Qni, and Sni).

3. Labelling patches as outliers if both thresholds in Equation (4.8) are reached (labelled MAD+, Qn+, and Sn+).

4. Same as above but also ignoring the outlier patches in the next frame to calculate the optimal set of candidate patches (labelled MADi+, Qni+, and Sni+).

The results of these experiments can be seen in Figure 4.8. Marking a patch

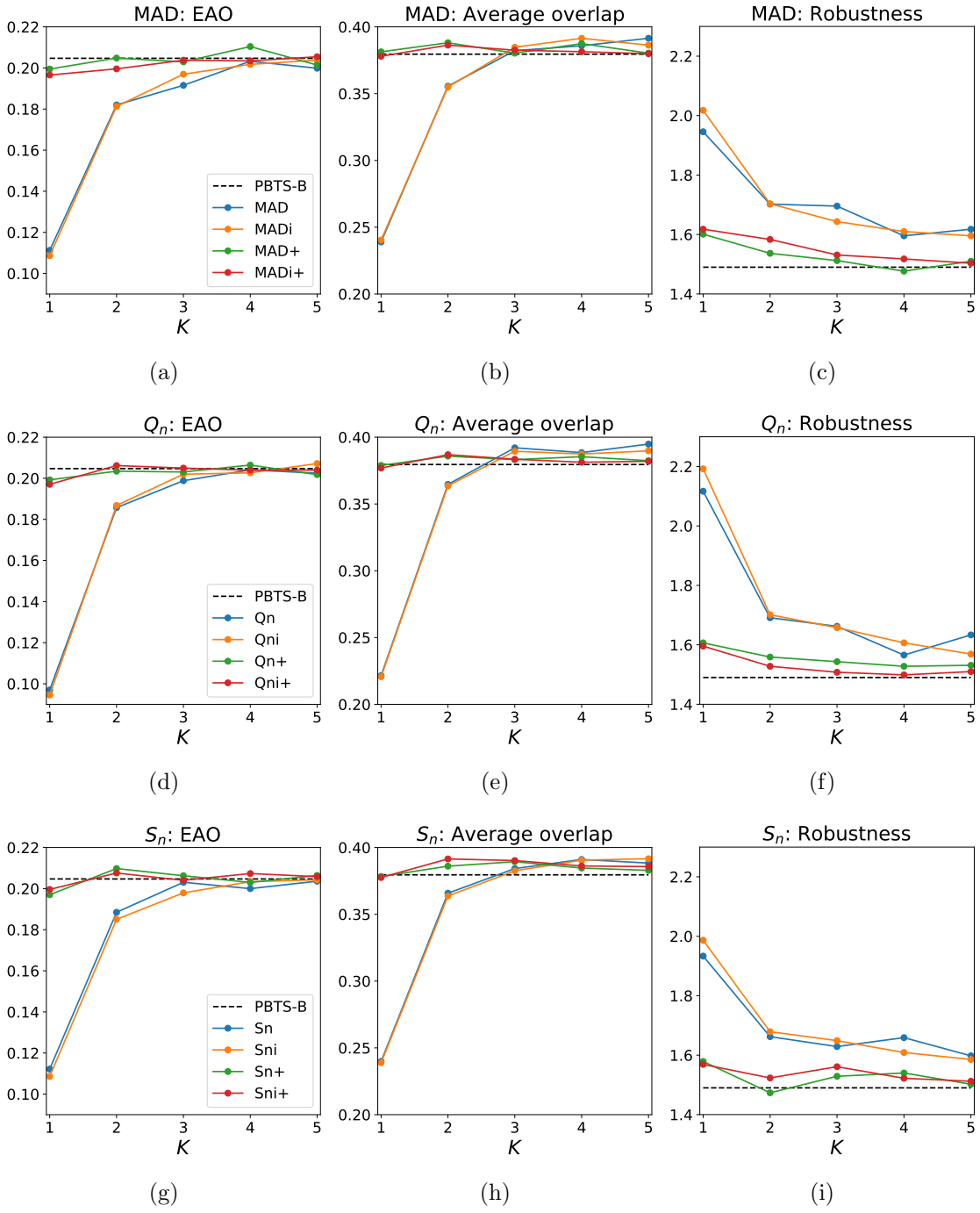


Figure 4.8: Tracking performance using outlier detection methods based on each patch's location relative to all other patch locations, compared to without using the method (PBTS-B, dashed). Each row shows the EAO, average overlap, and robustness for the three methods, MAD (a-c), Q_n (d-f), and S_n (g-i). The suffix i corresponds to ignoring the patch when predicting the object's bounding box if the patch was labelled as an outlier in the previous frame. The suffix + refers to labelling a patch as being an outlier only if it is marked as being an outlier in terms of both its horizontal and vertical location.

as being an outlier if either its horizontal or vertical location, relative to all others, indicates that it is an outlier, gives much worse performance for all three detection methods when using lower values of K, as can be seen from Figures 4.8a, 4.8d and 4.8g. This is somewhat expected because having a low threshold when only evaluating one direction would mean that any patches that drifted away from the central group of patches would be marked as an outlier very quickly, even if they are still within the object's bounds. This would result in predicting a smaller bounding box than would have otherwise been predicted and therefore would generate a smaller overlap with the object's actual location, as reflected in the average overlap scores (e.g. Figure 4.8e). For $K \geq 4$ the performance (EAO) of the methods become approximately equal to not using them. However, looking at the average overlap (e.g. Figure 4.8h) and robustness scores (e.g. Figure 4.8i) shows that the methods give a slight increase in accuracy at the expense of robustness. We suggest that this may be because the bounding boxes being predicted are smaller and would therefore result in a smaller search area in subsequent frames, which may lead to an insufficient area being searched in order to locate the object. This could potentially be counteracted by increasing σ_x and σ_y (the search scale parameters, discussed in Section 3.4.6), allowing for a larger search region to be evaluated.

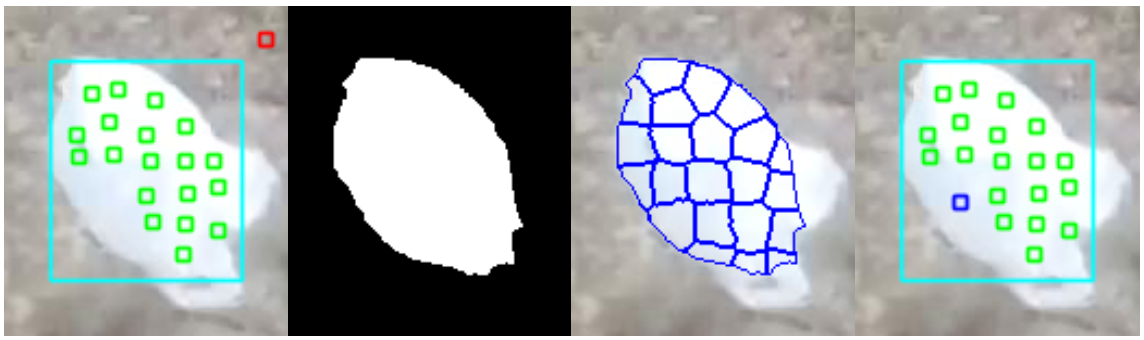
The methods that mark patches as outliers when both its horizontal and vertical locations indicate that it is an outlier (those methods suffixed with +) performs approximately the same as not using any outlier detection (e.g. Figure 4.8g) for all values of K. This is a surprising result as one would expect that if a patch is an outlier in both its horizontal and vertical location then ignoring it would certainly improve performance. However, we suggest that this lack of improvement may be due to patches rarely drifting away in two directions at the same time. This is reflected by the performance of the tracker when discarding patches when predicting boxes using a threshold of $K = 1$. It is also interesting to note that all three methods perform approximately equally, which suggests that the differences between the scales estimated for each method are less than the different threshold values (i.e. differences in scale are less than 1).

All three outlier detection methods, failed to improve on the PBTS-B tracker in terms of EAO, although they were able to achieve a marginally better average overlap at the expense of robustness. These results reflect the difficulty in accurately identifying patch outliers. We suspect that for an identical configuration of patches on two different objects that different patches should be marked as being outliers depending on the object in question. This naturally raises the question as to what other information within the image can be used to guide the outlier detection processes. Some trackers in the literature (Section 2.1.3), such as DPCF (Akin et al., 2016) use an secondary, global, tracker to model the object's entire bounding box. This is one potential avenue for future investigation as it would allow patches that drifting outside of this region to be marked as being an outlier.

Given that no patch outlier detection method based on location gave any statistically significant improvement when compared to the PBTS-B tracker, we choose not to include them directly in the final tracker. We do, however, select the best performing set of parameters for each method, S_n ($K = 4$), MQT ($T = 0.15$), and S_n+ ($K = 2$), and combine them with the patch replacement strategy in the following section. This is carried out in order to assess whether replacing the patches marked as being an outlier improves the performance of the methods.

4.4.3 Patch Replacement

If a patch has been labelled as being an outlier, i.e. it is not contributing well to tracking performance due to its poor match quality or its location, a common approach (e.g. LGT (Čehovin, Kristan and Leonardis, 2013), BHMC (Kwon and Lee, 2009), and RPAC (Liu, Wang and Yang, 2015)) is to remove the patch (and its model) from the object's set of patch models and replace it with a new one. The question of where the new patch should be placed is a non-trivial one, as the only regions of the image that we have any certainty of belonging to the object are those tracked with the non-outlier patches. This problem is very similar to the initialisation problem (Chapter 5) as we also need to make an informed choice about which pixels belong to the object, given limited information. In this section we describe our part placement



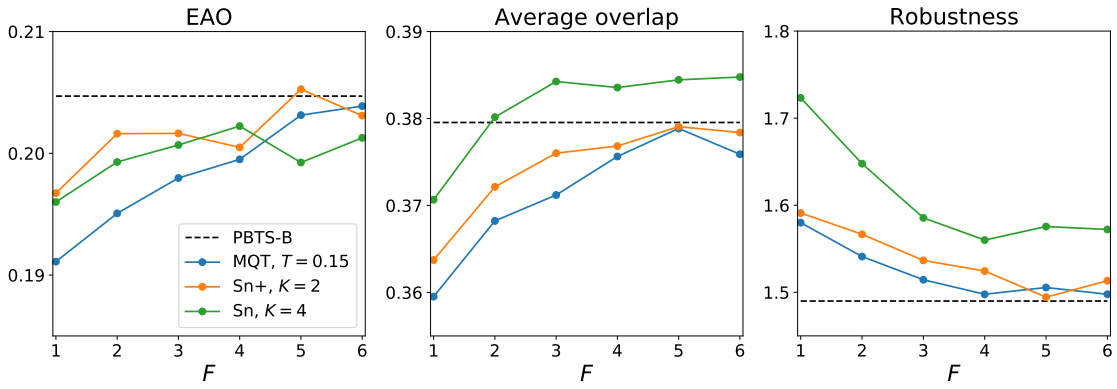
(a) (b) (c) (d)

Figure 4.9: The patch replacement scheme outline. Given that one or more patches have been labelled as an outlier (red patch in a) that needs replacing, we segment the axis-aligned bounding box (cyan) based on the non-outlying patches (green) using the alpha matting method from Section 5.1.3 (b). This region is then superpixeled (blue) (c) and a new location for each patch is selected at the centre of superpixels, resulting in new patches (blue) added to the model (d).

scheme in which we use our object part placement method (Section 4.1.1) to select new locations for the patches, create new patch models for them (Section 4.2.1), and then replace the outlying patches with the newly placed patches.

After one or more patches have been labelled as being an outlier (e.g. the red patch in Figure 4.9a), we generate an axis-aligned bounding box fitted to the non-outlying patches and expand it 20% vertically and horizontally. This method is identical to our object prediction scheme in Section 3.4.1. We then adapt the initial patch placement algorithm, described in Section 4.1.1, to take into account the position of the non-outlying patches. This starts by segmenting the region defined by the axis-aligned bounding box, using the alpha matting object segmentation scheme to be discussed in Section 5.1.3, which returns a mask predicting the location of the object (Figure 4.9b). This region is then superpixeled (Figure 4.9c) and centroids of superpixels which do not already contain any patches are used as potential patch locations. Starting with the largest superpixel, new patch locations are selected so that they can only overlap existing patches in the model by a proportion less than γ of its area. Once a new patch location (blue in Figure 4.9d) has been selected, a model is generated for it (Section 4.2.1, Algorithm 5).

In the following experiment we combine the aforementioned replacement scheme with the best-performing likelihood thresholding method, MQT ($T = 0.15$) and the two best location thresholding methods, Sn ($K = 4$) and Sn+ ($K = 2$). An



(a) (b) (c)

Figure 4.10: Tracking performance when replacing patches that have been identified as being outliers for F consecutive frames. MQT labels patches as outliers if their match quality drops below $T = 0.15$. Sn+ labels patches as outliers if they are more than $K = 2$ scale factors away from the median horizontal and vertical locations of the patches. Sn labels patches as outliers if they are more than $K = 4$ scale factors away from the median patch horizontal or vertical patch location. The dashed line represents the performance of the tracker without either modification.

additional criterion is also included for replacing a patch: a patch is replaced only if it has been labelled as an outlier for a certain number (F) of consecutive frames. This is included to assess whether immediately replacing patches is a good strategy, or whether delaying to ensure that the patch continues to be an outlier is preferred. We evaluate replacing the patches, after being labelled an outlier, in $F \in \{1, 2, 3, 4, 5, 6\}$ consecutive frames, e.g. $F = 1$ corresponds to replacing patches as soon as they are detected as being an outlier.

Figure 4.10 shows the results of the three methods, MQT (blue), Sn+ (orange) and Sn (green), compared to using no outlier detection or patch replacement (dashed). The figures show that, using these three methods to identify patches that are drifting and replace them after F frames does not appear to improve performance. The average overlap (Figure 4.10b) and robustness (Figure 4.10c) graphs suggest that increasing the number of consecutive frames needed for a patch to labelled as an outlier in order to be replaced brings the performance closer to that of vanilla PBTS-B, i.e. not using any outlier detection or replacement. We suggest that this is because as F increases then so too does the likelihood of a patch no longer being labelled an outlier. This may be due to one of two reasons. The first is that the optimisation scheme tries to move the patch back towards an area of higher match

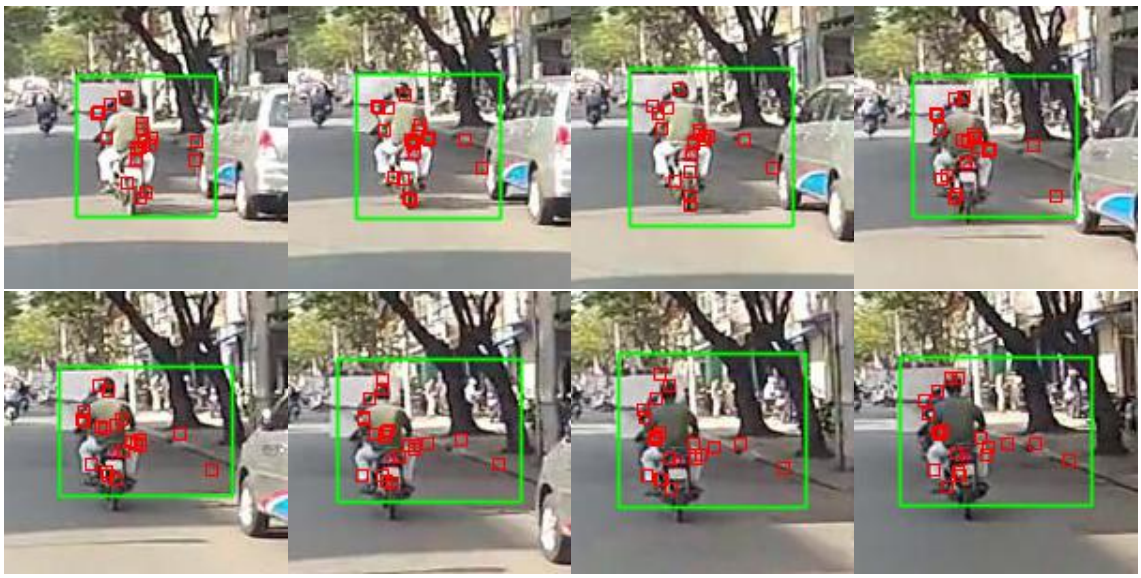


Figure 4.11: The `traffic` sequence from the VOT2016 dataset being tracked using the Sn+ ($K = 2$) outlier detection method combined with replacing patches that are labelled as outliers for $F = 5$ consecutive frames. The object's predicted bounding box is shown in green, with the tracked patches in red. The tracked patches on the right-hand side of the bounding box are slowly moving away from the majority of the patches and begin to track the background. Drift detection has failed in this case because there are several patches drifting at once, resulting in the scale parameter of the patch location distributions being overestimated.

quality, which will most likely be towards the other patches in the model. The second potential explanation is that the object's size may have been predicted to have reduced, resulting in patches becoming closer together, thereby bringing the patch back closer to the object for a higher match quality and also potentially allowing for the local optimisation process to move it towards the object.

It is clear that all forms of outlier detection and patch replacement in these sets of experiments yield worse overall performance, or at best equal performance, when compared with not using the methods. An example of the outlier detection method Sn+ ($K = 2$, $F = 5$) failing to detect patch drift can be seen in Figure 4.11, in which several patches have drifted away from the motorcycle rider and are tracking the road next to them. This has most likely occurred because the patches are only drifting horizontally, however the same situation would likely still occur using the Sn method (i.e. labelling as an outlier if either vertical or horizontal drift occurs). This is because there are several patches not far from one another, meaning that a small K threshold would be needed in order to label them as outliers. However, as shown in the experiments in the previous section (Figure 4.8), smaller thresholds

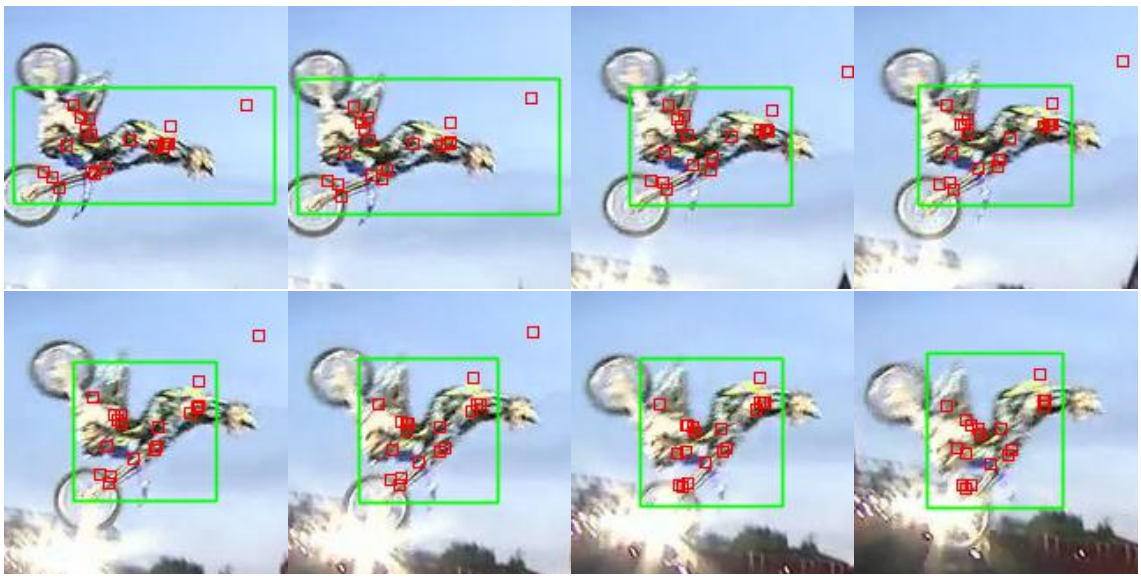


Figure 4.12: The `motocross1` sequence being tracked using the S_n+ ($K = 2$) outlier detection method combined with replacing patches that are labelled as outliers for $F = 5$ consecutive frames. The object's predicted bounding box is shown in green, with the tracked patches in red. Note that the outlier patch (upper-right) is detected in the third frame (hence not included in the predicted bounding box), and is replaced five frames later after being continually identified as an outlier.

too aggressively label patches as being outliers, resulting in poor performance. We emphasise that the methods can work well, as shown in Figure 4.12 where the motocross rider is being tracked. The sequence of images show a patch tracking the sky (rather than the rider) relatively far away from the other patches, and, after five frames of being detected as an outlier, it being successfully replaced with a new patch located on the motocross rider.

It is somewhat surprising that treating patch drift detection as an outlier detection problem gave no improvement to overall tacking performance, and at best gave equal performance to not including a detection scheme. This apparent difficulty is reflected in the literature (see Section 2.1.3 for a review of this), because most part-based trackers do not explicitly try to address drift. Instead they try to enforce constraints to make it very unlikely, or impossible, for patches to drift. These include using a separate model likelihood term that is based on the spatial relationship between patches, such using as a spring-like system to constrain a patch's movement relative to its neighbours (Yi et al., 2015), or using modelling the patches' relationship to one another using graphs and optimising these to enforce spatial consistency between object parts over multiple frames (Du et al., 2017).

In our tracking method, an individual patch is able to move away from other patches at a rate of up to two pixels per frame due to local optimisation (using a window size of $W = 5$ in Algorithm 3). They can also move more pixels per frame due to the global affine transformation predicting a scale increase, although this will not affect the relative distance between patches. Even though only small patch movements per frame are permitted, these can lead to large drifts over multiple frames. One avenue for future work may involve analysing the trajectories of patches over multiple frames to investigate how patches move in comparison to both their neighbours and the object as a whole. If a patch is continually moving in a way that is different from its neighbours or the average trajectory of the object, such as if a patch is tracking the background rather than an object, then this should indicate that the patch is likely to be an outlier.

4.5 Summary and Conclusion

In this chapter we have proposed methods for selecting where to place object parts in the first frame of video (Section 4.1), methods for selecting which pixels to include in a patch's model (Section 4.2), and introduced the patch model update scheme (Section 4.3). We also performed investigated patch drift, proposed several methods to identify drift patches, and described a method for replacing patches that have been identified as drifting (Section 4.4).

The problem of selecting where to place patches in the first frame of a video sequence, given a bounding box indicating approximately where the object is located, is a challenging one. This is because not all of the pixels within the given bounding box belong to the object and that selecting patch locations that lie over background regions may result in an object model having conflicting objectives because some parts will be trying to track the background and others will be trying to the track the object. This was addressed by first using the alpha matting method we developed in Section 5.1.3 to segment the object within the bounding box from its background and then superpixeling the image and placing patches at the centre of the superpixels.

Placing patches at the centre of superpixels was compared with placing them

at the boundaries of superpixels, preferring to place patches at locations that would overlap the most superpixels. This was carried out to compare whether placing patches in homogeneous regions (centres of superpixels) gave higher tracking performance than in inhomogeneous regions (edges of superpixels). While it might be expected that inhomogeneous regions would be easier to identify and unambiguously track, it was found that placing object parts at the centre of superpixels gave consistently better performance across all three metrics (EAO, average overlap, and robustness). We suggest that this is due to homogeneous regions being larger, and therefore less susceptible to blur or changes in illumination. If part of a homogeneous region changes then a patch is still able to move, due to the combination of the affine transformation and local optimisation, towards the region that still has the most similar appearance. Whereas the regions that overlap superpixels are naturally limited by the patch's size and so image changes have a more profound affect because even if only part of the region changes then the patch's model can, at best, only match the unchanged region, resulting in a lower match quality.

One avenue for future work is to investigate the performance benefits of using the patch location selection method (Algorithm 4) for other part-based trackers. As it is model-agnostic and only uses a frame of video and a given bounding box, it would be suitable for other part-based techniques. We suspect that it would improve the performance of several trackers, particularly those that place their patches in a grid-like formation, such as those discussed in Section 2.1.3 e.g. LGT (Čehovin, Kristan and Leonardis, 2013), LPT (Yi et al., 2015), and DCCO (Johnander et al., 2017). Their performance should improve as they would start with patches most likely placed on the object, rather than some patches being placed partially or completely on the object's background.

We also compared the amount of overlap allowed between patches when their locations are first selected on the object, thus controlling the amount of repeated information. It was found that having some amount of overlap (25% of a patch's area) was preferable to having none, and that having a large overlap led to poorer performance. We posit that this is due to two factors, firstly, allowing some amount

of patch overlap increases the number of patches that can be fitted onto smaller objects. Secondly, allowing for patches to overlap too much might result in a loss of descriptive power as more of the same colour information will be included in the model, rather than selecting patch locations that are further away from one another and therefore more likely to include different colour information.

Once a patch location has been selected, our initialisation scheme creates its model. It does this by randomly selecting pixel's features that are sufficiently away from any others in the model and uses them as the centres of spherical bins (similar to a normal histogram) in colour space. Counts how many pixels match each feature sample are also recorded, resulting in a set of feature-count pairs that empirically characterise the colour space in which the patch resides. We compared the random selection of feature samples to two, deterministic, selection schemes based on covering the least (i.e. compact) or most (i.e. expansive) colour space.

It was found that there was not much difference in performance between the three methods, although the expansive representation did perform slightly worse. We suggest, because of the homogeneous nature of the regions of the image that the patches are modelling (i.e. the centre of superpixels), that the specific sampling scheme is somewhat unimportant. The slight decrease in performance for the expansive representation suggests that it is taking up too much colour space and is consequentially is matching too much extra colour information that would not be matched by the more compact representation. This is also reflected in the expansive representation's lower robustness score, as it is worse than the other two methods but comparatively equal in average overlap.

Given the optimal set of patch locations, our model update scheme employs a two-pronged approach to update a patch's model. It moves feature samples towards the mean of the features of pixels that matched it in the patch's new location, as well as updating its count of how many pixels matched it. This is carried out using two separate update rates. The use of two update rates is necessary as the rate at which the relative proportions of the colours in the model should change at a different rate as the colours in the model itself. We evaluated a large range of values for both the

count update rate and the feature update rates. It was found that the optimal count update rate was small, with a value of 0.05, meaning that the number of matches a feature has will have to change substantially and often for the model to change significantly. The optimal feature update rate, the rate at which the spherical bin centre was moved towards the mean of the features that matched it, was found to be a rate of 1.7, i.e. a predictive rate that moved it past the mean of the features that it matched. This is an interesting result as it suggests that the colours of an object are continually changing and it is important to adapt to that as quickly as possible.

Our update scheme contrasts with the traditional histogram update scheme that only updates its bin counts as its bin centres are fixed and predetermined by the number of bins used to discretise the feature space. This causes problems for histogram-based approaches because if the colours being matched into one bin slowly drift towards another bin, e.g. a pixel's intensity slowly increases, the matches will eventually fall into another bin. This will result in the histogram patch's new histogram matching its model poorly (when using the Bhattacharyya distance) as the probability mass will be located in a neighbouring bin[2]. Given that the optimal update rate for features was found to be far greater than 0, this indicates that the ability of the patch model's features (i.e. bin centres) to move towards the area of colour space occupied by its matches is an important quality. This is also reflected in the experiments carried out in Section 3.3.3 in which our model was found to out-perform histogram-based models.

Lastly, we investigated how to limit the influence of drifting patches in our tracker. This was carried out by evaluating two types of schemes to identify drifting patches and label them as outliers not to be included in the creation of the bounding box used to report the object's location. These involved identifying outliers based on a patch's match quality and also its location relative to other patches. The match quality-based approach, which labelled patches as being outliers if they had a match quality below a certain threshold, did not lead to an increase in tracking performance. In fact, once the threshold was sufficiently large, it led to a decrease in tracking

[2] This could be avoided by using the Earth Mover's Distance (Rubner, Tomasi and Guibas, 1998), but this has a much larger computational cost.

performance. We suggest that this indicates that a patch's match quality may not be a good indicator of whether a patch is drifting, because a patch could just be matching poorly due to the object changing in appearance and will recover over several frames as its model is updated.

Three methods based on estimating the scale parameter of the distribution of the both the vertical and horizontal locations of patches were also investigated. These label patches as being outliers if their standardised location was more than a certain multiple of the scale parameter away from the centre of the patches. We compared labelling a patch as being an outlier if they were an outlier in either their vertical or horizontal location, to labelling a patch as an outlier if they were an outlier in both horizontal and vertical location. It was found that, even for a small threshold, labelling patches that were an outlier in both directions as an outlier did not give any significant performance increase. Similarly the alternative method, which required a larger threshold to achieve the same performance, also did not provide any average improvement over not using any drift detection, despite its efficacy in some situations.

In an attempt to improve the outlier detection process, we developed a patch replacement scheme that replaces patches after being labelled as an outlier in several consecutive frames. This is similar to the patch placement scheme outlined previously, and started by segmenting the object inside an axis-aligned bounding box that surrounded the patches that were not marked as outliers. The image was then superpixeled and new patch locations were selected by choosing the centres of superpixels that were not already occupied by a patch in the model and so that the new patch only overlaps current patches by a certain proportion. Once the locations were selected a patch model was created using the model creation scheme previously discussed. We combined the replacement scheme with the best performing outlier detection methods and evaluated their performance.

It was found that using this scheme made little performance improvement in conjunction with any of the outlier detection schemes evaluated, even with a delay before making a patch labelled as an outlier for replacement. As the delay

Component	Value
Patch placement method	Superpixel centroids
Patch overlap amount (γ)	0.25
Features update rate (β_s)	1.7
Counts update rate (β_c)	0.05
Patch drift detection method	None
Patch replacement method	None

Table 4.3: The optimal tracking components and their attributes selected in this chapter.

increased, the performance of the trackers approached that of not using any patch replacement and outlier detection. We note, that the average overlap (i.e. the accuracy of the tracker) improved slightly when using horizontal or vertical outlier detection. However, this was counteracted by being less robust, resulting in a very similar EAO to not using the method. We suggest that this is because different objects need different scale parameter thresholds in order to correctly label outliers, something which we have not carried out in this work. This is a difficult problem because there may not be a straightforward relationship between the size of object and its corresponding optimal scale parameter threshold. Therefore we leave this as a possible topic for future work. We note again that our method, like others, only uses information in the current frame to determine if a patch is an outlier. It may be advantageous to compare the trajectory of a patch over multiple frames to the other patches in the object model. We suspect that, in general, the trajectory of a drifting patch would be significantly different to a non-drifting patch. This is because drifting patches tend to be tracking the object's background, which should remain relatively stationary in comparison to the object.

For completeness, in Table 4.3 we summarise the optimal components and their parameters that are used in the final tracker, selected according on the experiments carried out in this chapter. So far, we have not discussed how the object to be tracked is detected in the given bounding box of the initial frame. In the following chapter, we therefore introduce the initialisation problem together with three methods to address it. They are empirically evaluated on the VOT2016 dataset and the best method is used to segment the bounding box during the initial patch placement in the first frame of video.

5. The Initialisation Problem

Recent benchmarks for object tracking have focused on the single-target, short-term, model-free tracking problem (Wu, Lim and Yang, 2013; Kristan et al., 2015). The model-free aspect of the problem is particularly challenging because this means that a tracker has no *a priori* knowledge of the characteristics (such as colour, shape, and texture) of the object. Tracking algorithms are only provided with the first frame of a video and a bounding box that indicates which region of the image contains the object. Typically up to 30% of this region can be contain pixels that do not belong to the object, i.e. background pixels (Vojíř and Matas, 2017). Initialising tracking algorithms with background instead of foreground can be severely deleterious to their performance, so in this chapter we examine the initialisation problem of locating the object to be tracked within the given bounding box. We refer to the problem of determining which pixels, given an approximate location of the object, belong to the object and which do not, as the *Initialisation Problem*.

We address the initialisation problem by treating it as a missing labels problem, as we can be sure that pixels sufficiently outside the bounding box do not belong to the object. The problem is challenging, however, because bounding boxes are generally small and the object itself may be quite similar in appearance to the background. In addition, the object to be tracked may extend outside the given bounding box (Vojíř and Matas, 2017). For example, in order to reduce the proportion of background pixels within a bounding box, the outstretched limb of a person to be tracked might be excluded from the initialising bounding box. This means that the distance away from a bounding box where pixels are certainly background may be different for each segmentation. We note too that distance from the bounding box is not a guarantee that a pixel or region's appearance is not similar to the appearance of the object to

be tracked; a common example is tracking a particular face in a crowd.

We evaluate the performance of three techniques that have been tailored to the Initialisation Problem. The first two, using a One-Class SVM and a novel Sample-Based Background Model, treat the problem as a one-class classification problem. They attempt to learn the characteristics of the background regions of the image; pixels within the bounding box are then compared to the background model to determine object pixels. The third technique is an adapted solution to the image matting problem. This models each pixel in the image as comprising proportions of both foreground and background colour, aiming to to learn these proportions. The performance of these three techniques is evaluated by assessing their segmentation on the VOT2016 dataset (Kristan et al., 2016a). We discuss their strengths and weaknesses and investigate the robustness to parameter settings of the learned matting method.

The chapter is organised as follows. In Section 5.1 we formulate the initialisation problem and present the three techniques to be empirically compared. The experimental procedure and results are presented in Sections 5.2 and 5.3, and are followed by concluding remarks in Section 5.4.

5.1 Problem Formulation

We aim to determine the label $k \in \{0, 1\}$ of each pixel in the image, where the two labels represent belonging to the background of the bounding box and the object itself, respectively. The image I containing the object consists of a set of pixels indexed by $\Omega = \{1, \ldots, n\}$, where n is the total number of pixels in the image. Given a subset of these pixels $\Omega_b \subset \Omega$ which belong to the background, such that $k_i = 0 \,\forall\, i \in \Omega_b$, then the problem can be formulated as learning the correct labels of the unlabelled pixels $\Omega_u = \Omega \setminus \Omega_b$.

Rather than treating each pixel $i \in \Omega$, or features extracted from them, as a data point, two of the methods we examine use a superpixel representation, which provides a richer description of small, approximately homogeneous, image regions. Superpixeling techniques over-segment the image into perceptually meaningful regions

(the pixels in a superpixel are generally uniform in colour and texture), while at the same time retaining the image structure, as superpixels tend to adhere to colour and shape boundaries. Superpixeling can also significantly reduce the computational complexity because groups of pixels are represented as a single entity. An image I is therefore comprised of a set of superpixels $\mathcal{S} = \{S_j\}_{j=1}^{N_{sp}}$, where N_{sp} is the number of superpixels, and we associate a vector of features $\mathbf{x}_j \in \mathbb{R}^d$, such as a histogram of its RGB values, with each superpixel.

5.1.1 One-Class SVM

In general, pixels belonging to the object are not known, but a large number of pixels or superpixels may be assumed to belong to the background. We therefore treat the initialisation problem as that of identifying pixels which are significantly different from the background. This can be accomplished by using a One-Class SVM (OC-SVM) to estimate the support of the given background pixels.

The goal of a OC-SVM (Schölkopf et al., 2001) is to learn the hyperplane $\mathbf{w} \in \mathcal{F}$ that produces the maximum separation between the origin and data points $\{\mathbf{x}_i\}_{i=1}^{N}$, $\mathbf{x}_i \in \mathcal{X}$ mapped to a suitable feature space $\mathcal{F}$. It uses an implicit function $\Phi(\cdot)$ to map the data into a dot product feature space Φ: $\mathcal{X} \mapsto \mathcal{F}$ using a kernel

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle. \tag{5.1}$$

The radial basis function kernel (Gaussian kernel),

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^2}, \tag{5.2}$$

is a popular choice, as this guarantees the existence of such a hyperplane (Schölkopf et al., 2001). The decision function $f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x}) - \rho)$, is used to determine whether an arbitrary input vector $\mathbf{x}$ belongs to the modelled class when $\mathbf{w} \cdot \Phi(\mathbf{x}) \geq \rho$.

In order to find $\mathbf{w}$ and ρ, the problem can be defined in its primal form:

$$\min_{\mathbf{w}\in\mathcal{F},\,\boldsymbol{\xi}\in\mathbb{R}^N,\,\rho\in\mathbb{R}} \quad \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \frac{1}{\nu N}\sum_i \xi_i \tag{5.3}$$

$$\text{subject to} \quad \mathbf{w}\cdot\Phi(\mathbf{x}_i) \geq \rho - \xi_i, \quad \xi_i \geq 0, \tag{5.4}$$

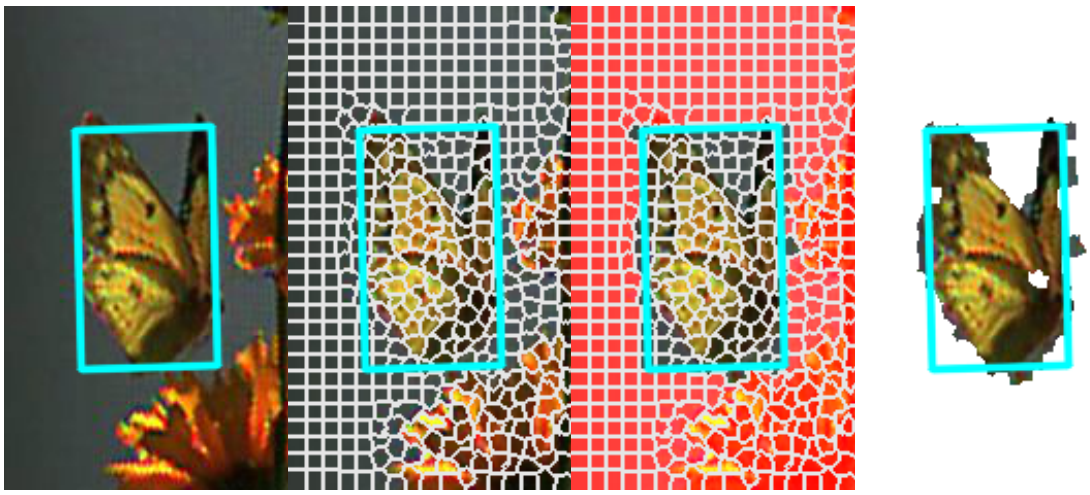
where slack variables ξ_i allow the corresponding $\mathbf{x}_i$ to lie on the other side of the decision boundary, and $\nu \in (0, 1)$ is a regularisation parameter. Conversion to the dual form reveals that the solution can be written as a sparse weighted sum of *support vectors* and solved by quadratic programming or, more typically, Sequential Minimal Optimisation (Chang and Lin, 2011).

The parameter ν acts both as an upper bound on the fraction of outliers (data points with $f(\mathbf{x}) < 0$) and a lower bound on the fraction of data points used as support vectors. It is worth noting that, for kernels such that $k(\mathbf{x}_i, \mathbf{x}_i) = c \,\forall\, \mathbf{x}_i \in \mathcal{X}$, where c is some constant, this formulation is equivalent (Schölkopf and Smola, 2001) to that of using Support Vectors for Data Description (Tax and Duin, 2004), in which the goal is to encapsulate most of the data in a hypersphere in $\mathcal{F}$.

Application

In order to use the OC-SVM approach, the image is preprocessed and features extracted in order to train the classifier and identify which regions of the image belong to the object. As illustrated in Figure 5.1a, the image I is first cropped to twice the width and height of an axis-aligned bounding box containing the supplied bounding box and object. The cropping allows for a sufficient region of the image to be included to train the classifier, while only including areas relatively close, in terms of the object's size, to the object. The cropped region is then segmented using a variant of the SLIC superpixeling algorithm (Achanta et al., 2012), SLIC0, which produces regular shaped (compact) superpixels (Figure 5.1b), while adhering to image boundaries in both textured and non-textured regions of the image.

The main parameter of SLIC0 is the approximate number of superpixels N_{sp} resulting from the segmentation. Following preliminary experiments, we aim to have an average of 50 pixels per superpixel to give a sufficient number of pixels from which



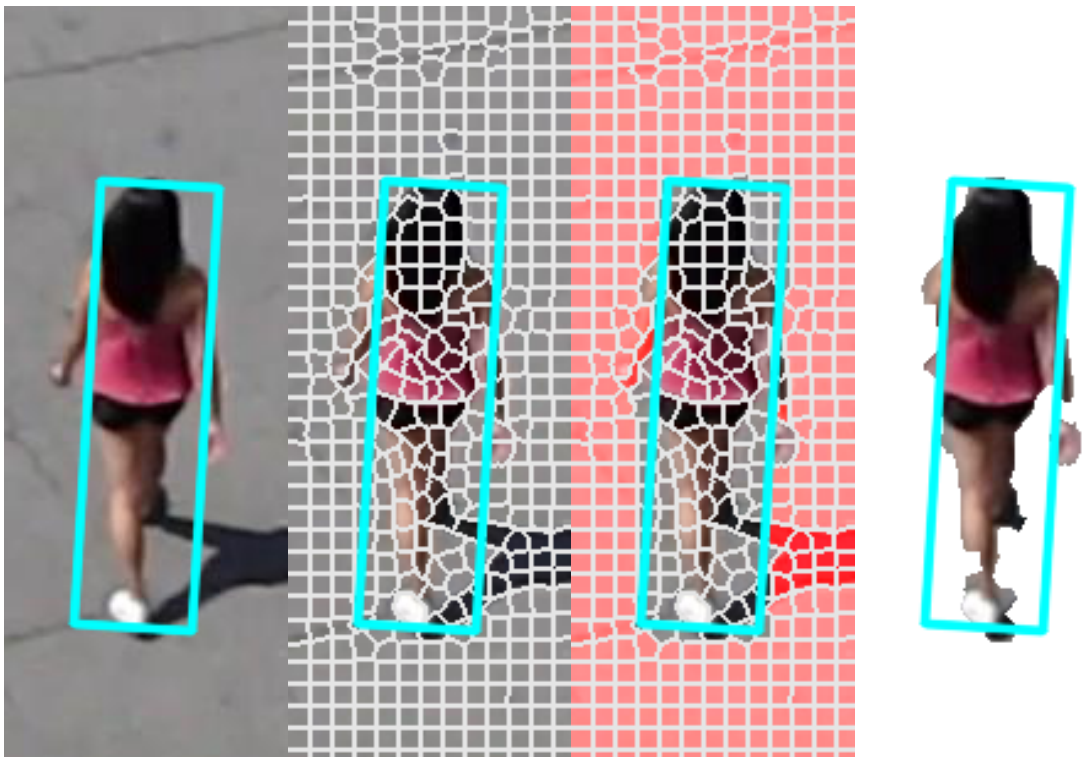
(a) (b) (c) (d)

Figure 5.1: The One-Class SVM object segmentation pipeline. The image is first cropped (a) and then superpixeled (b). An OC-SVM is then trained ($\gamma = 2^{-20}$, $\nu = 0.3$) on the features extracted from superpixels considered background (shaded red) in (c). Those with an unknown label (unshaded) have their class predicted, giving a final predicted object mask (d).

to extract features. However, given the varying size of each cropped image, this could result in very few or very many superpixels. Therefore we constrain the number of superpixels to lie in the range $\left[N_{sp}^{-}, N_{sp}^{+}\right]$. We use $N_{sp}^{-} = 100$ and $N_{sp}^{+} = 500$, as these have empirically given reasonable-looking segmentations and are similar to the range used by Du et al. (2017).

Following segmentation, superpixels that lie wholly outside the bounding box are labelled as not belonging to the object, such as the red superpixels in Figure 5.1c, with the features extracted from these forming the training data. We examine the performance of four alternative feature representations: histograms of RGB or, perceptually uniform, LAB pixel intensities across the superpixel; or the concatenated SIFT (Lowe, 1999) or LBP (Ojala, Pietikäinen and Mäenpää, 2002) feature vectors resulting from the R, G and B channels at the centroid of the superpixels. Both SIFT and LBP are popular texture-based feature descriptors, with SIFT features extracted based on the region's gradient magnitude and orientation, and LBP features formed based on differences in intensity between a pixel and its surrounding neighbours.

The classifier, with its parameters γ and ν, representing the kernel's length-scale and the classifier's upper bound on the assumed number of outliers in the training set, is subsequently trained on these data. Finally, features extracted from



(a) (b) (c) (d)

Figure 5.2: The Sample-Based Background Model object segmentation pipeline. The image is cropped (a) and superpixeled (b). A model ($\delta = 0.5$) is created for each superpixel (shaded red) in (c) outside the bounding box, and the other superpixels (unshaded) are compared to them ($\eta = 0.8$) and classified (d).

the unknown superpixels, lying wholly or partially within the given bounding box are classified with the trained OC-SVM. Figure 5.1d shows the resulting segmentation.

5.1.2 Sample-Based Background Model

Our sample-based one-class modelling technique is inspired by the ViBe background subtraction algorithm of Barnich and Van Droogenbroeck (2011). We represent labelled regions of the image, in this case superpixels, by sets of samples that characterise the colour distribution of these regions. In the same way as described in Section 5.1.1, the region surrounding the supplied bounding box is cropped and over-segmented into superpixels (Figures 5.2a and 5.2b). Superpixels lying wholly outside the specified bounding box are labelled as background (Figure 5.2c).

An unlabelled superpixel is compared to the modelled superpixels by evaluating how similar its set of samples is to each of the labelled sets of samples. The unlabelled superpixel is classified as either belonging to the labelled region (background) of the

image if it is sufficiently similar to any of the labelled sets of samples, or the object if not. We denote the model of the j-th superpixel S_j to be $m_j = \{\mathbf{x}_i\}_{i=1}^{s}$, consisting of a set of s pixel values $\mathbf{x}_i$ randomly sampled (with replacement) from S_j. As the average number of pixels in a superpixel, $\bar{N}_p$, may vary from image to image, we set $s = \delta \bar{N}_p$, where $\delta \in (0, 1]$ is chosen by cross-validation.

Let $\mathcal{M} = \{m_j\}$ be the set of models that characterise the superpixels which are located completely outside the bounding box. Then pixel $\mathbf{x}_p$ in a superpixel whose model is $m' = \{\mathbf{x}_p\}_{p=1}^{s}$, is deemed to match the model m_j if it is closer than a radius R to any pixel in m_j. Thus

$$Q\left(\mathbf{x}_p, m_j\right) = \begin{cases} 1 & \exists\, \mathbf{x}_i \in m_j : \left\|\mathbf{x}_i - \mathbf{x}_p\right\|_2 < R \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

The parameter R controls the radius of a sphere centred on each of the model's pixels, allowing for inexact matches. This is needed as lighting conditions usually vary within an image, so it permits matching pixels with subtle discrepancies in colour which would otherwise result in a mismatch.

The extent to which pixels in m' match m_j is then assessed by

$$q(m', m_j) = \frac{1}{S} \sum_{\mathbf{x}_p \in m'} Q\left(\mathbf{x}_p, m_j\right). \tag{5.6}$$

If $q(m', m_j) > \eta$ for any $m_j \in \mathcal{M}$ then the unlabelled superpixel is marked as matching a background superpixel and is deemed to be itself a background superpixel. If m' fails to match any background superpixel it is counted as foreground. Comparison of $q(m', m_j)$ with $\eta \in [0, 1]$ allows for some false positive matches before the superpixel is counted as background.

Figure 5.2 illustrates the pipeline. We remark that SBBM has correctly identified the majority of the shadows within the bounding box as background, and the person's shoes as foreground. However, parts of their arms and legs have been labelled as background because they match limb or shadow pixels outside the bounding box.

5.1.3 Learning Based Digital Matting

Digital matting, also known as natural image matting (Zheng and Kambhamettu, 2009), is the process of separating an image I into a foreground F and background B image, along with an opacity mask α. The colour of the i-th pixel is assumed to be a combination of a corresponding foreground and background colour, blended linearly,

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i, \tag{5.7}$$

where $\alpha_i \in [0, 1]$ is the pixel's foreground opacity. Solving for α_i, F_i, and B_i is extremely under-constrained as there are more unknown variables (F, B, α) than known variables (I). This ill-posed problem is made more tractable by supplying additional information, using either trimaps, where the vast majority of the image is labelled as being all foreground or all background pixels (e.g. Chuang et al. 2001), or by using scribbles, where only small regions, denoted by user-supplied scribbles, are selected as being foreground or background.

Learning Based Digital Matting (Zheng and Kambhamettu, 2009) trains a local alpha-colour model for all pixels Ω based on their most-similar neighbouring pixels. More specifically, any pixel's $i \in \Omega$ corresponding alpha matte value α_i is predicted via a linear combination of the alpha values $\{\alpha_j\}_{j \in \mathcal{N}_i}$ of its neighbours, where $\mathcal{N}_i \subset \Omega$ are the neighbouring pixels of i. Defining $\boldsymbol{\alpha}_i = [\alpha_{\tau_1}, \ldots, \alpha_{\tau_j}, \ldots, \alpha_{\tau_m}]^T$, with $\tau_j \in \mathcal{N}_i$ and $m = |\mathcal{N}_i|$, to be the α values of the neighbouring pixels, the linear combination is expressed as

$$\alpha_i = \mathbf{f}_i^T \boldsymbol{\alpha}_i, \tag{5.8}$$

where $\mathbf{f}_i = [f_{i\tau_1}, \ldots, f_{i\tau_j}, \ldots, f_{i\tau_m}]^T$ is the coefficients of the alpha values.

Equation (5.8) can also be written in terms of a linear combination of the alpha values for the entire image: $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^T$. Defining the $n \times n$ matrix $\mathbf{F}$ such that the i-th column contains the coefficients $\mathbf{f}_i$ in positions corresponding to $\mathcal{N}_i$, Equation (5.8) can be written as

$$\boldsymbol{\alpha} = \mathbf{F}^T \boldsymbol{\alpha}. \tag{5.9}$$

Assuming that $\mathbf{F}$ is known, $\boldsymbol{\alpha}$ can be estimated via a mean squared error minimisation

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \left\| \boldsymbol{\alpha} - \mathbf{F}^T \boldsymbol{\alpha} \right\|^2 + c \left\| \boldsymbol{\alpha}_b - \boldsymbol{\alpha}_b^* \right\|^2, \tag{5.10}$$

where $\boldsymbol{\alpha}_b \subset \boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_b^*$ are vectors of the predicted alpha values of the labelled pixels and the actual alpha values of labelled pixels respectively. The parameter c denotes the size of the penalty applied for predicting alpha values that are different from the user-specified labels; Zheng and Kambhamettu (2009) set $c = \infty$ in order to penalise any deviation and to maximally use the additional information provided by the known labels.

Equation (5.10) can also be written as

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha} \in \mathbb{R}^n}{\operatorname{argmin}} \boldsymbol{\alpha}^T \left(\mathbf{I} - \mathbf{F} \right) \left(\mathbf{I} - \mathbf{F} \right)^T \boldsymbol{\alpha} + \left(\boldsymbol{\alpha} - \hat{\boldsymbol{\alpha}} \right)^T \mathbf{C} \left(\boldsymbol{\alpha} - \hat{\boldsymbol{\alpha}} \right), \tag{5.11}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{C}$ is a diagonal matrix with $C_{ii} = c$ if $i \in \Omega_b$ and 0 otherwise, and $\hat{\boldsymbol{\alpha}}$ is the vector whose elements are the provided alpha values for $i \in \Omega_b$ and zero otherwise. Assuming $\mathbf{F}$ is known, this is solved by

$$\boldsymbol{\alpha}^* = \left(\left(\mathbf{I} - \mathbf{F} \right) \left(\mathbf{I} - \mathbf{F} \right)^T + \mathbf{C} \right)^{-1} \mathbf{C} \hat{\boldsymbol{\alpha}}. \tag{5.12}$$

The values in each column of $\mathbf{F}$ are computed via a local learning model to predict the value of α_i, in which a local colour model for each pixel $i \in \Omega$ is trained to describe the relationship between the data points in the pixel's neighbourhood and their alpha values. A linear alpha colour model is chosen for the local learning process:

$$\alpha_i = \mathbf{x}_i^T \boldsymbol{\beta} + \beta_0 = \mathbf{x}_i'^T \boldsymbol{\beta}', \tag{5.13}$$

where $\mathbf{x}_i' = [\mathbf{x}^T 1]^T$, $\mathbf{x}_i$ is the i-th data point (in this case an RGB vector), and $\boldsymbol{\beta}' = [\beta_1, \cdots, \beta_d, \beta_0]$ are the model coefficients. For the sake of readability, we drop the primed notation for both $\boldsymbol{\beta}'$ and $\mathbf{x}'$ and use $\boldsymbol{\beta}$ and $\mathbf{x}$ to mean the model coefficients and data point respectively. If $\mathbf{X}_i = [\mathbf{x}_{\tau_1} \cdots \mathbf{x}_{\tau_m}]^T$ is a matrix populated by the data points in the neighbourhood of pixel i, Tikhonov regularisation (ridge regression)

allows $\boldsymbol{\beta}$ to be estimated (Zheng and Kambhamettu, 2009) by solving

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\| \alpha_i - \mathbf{X}_i \boldsymbol{\beta} \right\|^2 + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}, \tag{5.14}$$

which has the solution

$$\boldsymbol{\beta}^* = \mathbf{X}_i^T \left(\mathbf{X}_i \mathbf{X}_i^T + \lambda \mathbf{I} \right)^{-1} \alpha_i. \tag{5.15}$$

Here λ is the shrinkage parameter of the regularisation that controls the size of the penalty placed on the regularisation coefficients. This can be substituted into Equation 5.13 and shows that the non-zero values of the i-th column of $\mathbf{F}$ are only dependent on the features of each pixel:

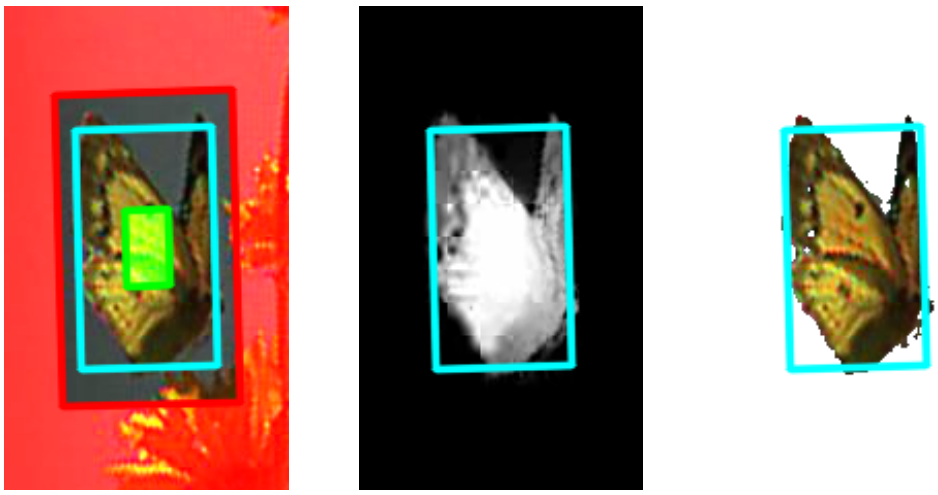
$$\mathbf{f}_i = \left(\mathbf{X}_i \mathbf{X}_i^T + \lambda \mathbf{I} \right)^{-1} \mathbf{X}_i \mathbf{x}_i. \tag{5.16}$$

Application

In order to apply this algorithm to the initialisation problem a *scribble mask* is needed. This contains labels for pixels that definitely belong to the background, the object, and the unknown region. In typical applications the scribble mask is provided via user input, however this is not possible within model-free object tracking, as no additional *a priori* information can be provided.

We have addressed this by automating the creation of a scribble mask based on only the bounding box and image, cropped in an identical manner to that of the superpixeling algorithms. The area of the original bounding box (cyan box in Figure 5.3a) is decreased by a factor of $\rho^- \in (0, 1)$, linearly shrinking the height and width of the bounding box by $\sqrt{\rho^-}$. The pixels within this region are labelled in the scribble mask as belonging to the object, shown by the green-shaded area in Figure 5.3a. Similarly, we also increase the area of the original bounding box by a factor of $\rho^+ \in (1, 2]$, linearly expanding both of its dimensions by $\sqrt{\rho^+}$. Pixels outside this region are labelled as belonging to the background (shaded red in Figure 5.3a), leaving pixels between the two labelled regions as being of unknown origin.

Shrinking the area of the bounding box and using this region as a scribble

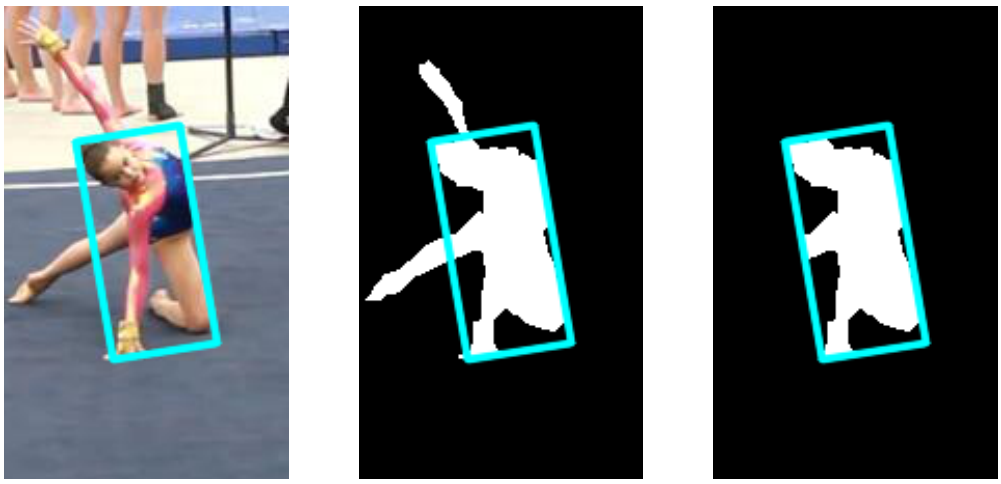


(a) (b) (c)

Figure 5.3: The Alpha matting object segmentation pipeline. The object's bounding box (cyan) is expanded (red, $\rho^+ = 1.7$) and all pixels outside this region are labelled as belonging to the background (shaded red). The bounding box is also contracted (green, $\rho^- = 0.9$) and pixels within this contracted bounding box are labelled as belonging to the object (shaded green) (a). The alpha matting method then learns the labels for the unlabelled pixels (i.e. those between the two shaded regions), creating an alpha mask (b). This mask is then thresholded ($\tau = 0.80$) and pixels with alpha values greater than the threshold are labelled (c) as belonging to the object.

mask to represent the object relies on the assumption that the object is located at the centre of the bounding box, which may not always be the case. If an object is highly non-compact or has holes in it, then there is a possibility of background pixels being included in the scribble mask. Expanding the bounding box to represent the background also makes the assumption that pixels sufficiently far away from the bounding box do not belong to the object. This is a stronger assumption, although there may still be cases where parts of an object protruding far from the bounding box are labelled as background.

The output of the alpha matting process, the alpha matte $\boldsymbol{\alpha}^*$ (Equation 5.12), is not a strict segmentation, but rather a pixel mask indicating what fraction of each pixel is foreground (belonging to the object) and background. A segmentation is produced by thresholding the alpha values to create an object mask. Pixels with a predicted α_i^* value greater than a threshold t are assigned as belonging to the object, with those whose $\alpha_i \leq t$ being labelled as background. The threshold t is chosen so that a proportion τ of the bounding box is classified as the object. This allows the alpha threshold t to be dynamically chosen based on how much of the



(a) (b) (c)

Figure 5.4: The segmentation criteria used, based on the original cropped image and bounding box (a). ϕ_{all} uses the full ground-truth segmentation mask of the object (b) and ϕ_{bb} uses the same mask limited to the given ground-truth bounding box (c).

bounding box is expected to be populated with the object in question. For example, in the VOT2016 benchmark (Kristan et al., 2016a) at least 70% of the bounding box belongs to the object.

5.2 Evaluation Protocol

In order to evaluate the segmentation performance of each algorithm, we used real-world input in the form of the VOT2016 dataset (Kristan et al., 2016a). It comprises 60 videos with human-annotated ground-truth bounding boxes and pixel-wise segmentations (Vojíř and Matas, 2017). In order to increase the number of examples, we used the first, middle, and last frames from each video. Within this dataset, the sizes of the bounding boxes varies widely. The minimum and maximum dimensions were 8.5 and 511 pixels respectively, with the mean dimension being 92 pixels. The number of pixels contained within the bounding boxes ranged from 229 to 214773 with the mean being 12183.

The segmentation performance was assessed by the IOU measure (Equation 2.3). Two criteria were used,

$$\phi_{all} = \frac{|\mathcal{G} \cap \mathcal{P}|}{|\mathcal{G} \cup \mathcal{P}|} \tag{5.17}$$

and

$$\phi_{bb} = \frac{|\mathcal{G} \cap \mathcal{P} \cap \mathcal{B}|}{|(\mathcal{G} \cup \mathcal{P}) \cap \mathcal{B}|}, \tag{5.18}$$

which compared the predicted object mask $\mathcal{P}$ to the ground-truth $\mathcal{G}$ mask in two different ways. As illustrated in Figure 5.4b, ϕ_{all} includes regions of $\mathcal{G}$ that lie outside the bounding box. This assesses the full segmentation capability of the technique, and allows a tracker to make use of all available information about the object and its background. The ϕ_{bb} measure compares the quality of object segmentation and the ground-truth within a given bounding box, $\mathcal{B}$. This corresponds to the assumption made by many tracking algorithms that the object lies completely within the bounding box and that it should only track pixels within it.

Each technique was evaluated using leave-one-video-out cross-validation, meaning that the 3 frames from a single video were held out for testing, while the optimal parameters for the technique were determined as those giving the best ϕ_{all} or ϕ_{bb} score averaged over the 177 frames from the remaining 59 videos. Performance on the 3 held out frames was then evaluated using the cross-validated optimal parameters. This procedure was repeated for each of the other 59 videos held out in turn to obtain average performance metrics.

Each possible combination of the parameter values shown in Table 5.1 was evaluated for each technique, with the process being repeated for both ϕ_{all} and ϕ_{bb}. In addition, a baseline performance, denoted as "Entire BB", was obtained as the

Technique	**Parameter**	**Values**
OC-SVM	ν	$0.001, 0.002, \ldots, 0.005,$ $0.01, 0.02, \ldots, 0.05,$ $0.1, 0.15, \ldots, 0.5$
	γ	$2^{-20}, 2^{-19}, \ldots, 2^{20}$
SBBM	δ	$0.1, 0.2, \ldots, 1.0$
	η	$0.1, 0.2, \ldots, 1.0$
LBDM	ρ^+	$1.1, 1.2, \ldots, 2.0$
	ρ^-	$0.1, 0.2, \ldots, 0.9$
	τ	$0.50, 0.51, \ldots, 1.00$
	λ	$10^0, 10^{-1}, \ldots, 10^{-10}$

Table 5.1: Parameter values evaluated for each technique during the leave-one-video-out cross-validation experiments. The techniques were evaluated using each possible combination of the parameter values shown.

segmentation that consists of the entire bounding box; that is $\mathcal{P} \equiv \mathcal{B}$.

As noted in Section 5.1.1, the OC-SVM is a feature-based classifier. Two colour-based features, RGB and LAB, as well as two texture-based features, SIFT and LBP were used for separate experiments within the same testing protocol. We used 8 bins for each channel in both the RGB and LAB feature descriptors to represent each superpixel. The SIFT features used were the concatenation of SIFT features extracted for each colour channel separately; likewise for LBP. Dense SIFT features were extracted with a 4×4 sliding window and with the orientation set to 0. The feature descriptor extracted nearest to a superpixel's centroid was assigned as its feature. LBP features from a 5×5 window were extracted, centred on the pixel nearest to the superpixel's centroid. We used the typical parameters of $P = 8$ and $R = 2$, with the scale and rotation invariant version of the descriptor, along with the uniform patterns extension (Ojala, Pietikäinen and Mäenpää, 2002). Features were standardised to have zero mean and unit variance before classification.

SBBM used the raw RGB values of the pixels and a colour radius $R = 20$, corresponding to approximately a 4.5% deviation in each colour channel, which we have found to give good results across a range of examples.

In the alpha matting technique, the neighbourhood $\mathcal{N}_i$ was defined to be a 3×3 region centred on the pixel in question, meaning that the size of each pixel's neighbourhood was $m = 8$. We used the RGB values to be the features $\mathbf{x}_i$ of each pixel, normalising them such that $[0, 255] \mapsto [0, 1]$. The penalisation for deviating from the given alpha matting label was set to $c = 800$ as this is a large enough penalty, compared with normalised values of $\mathbf{x}_i$, to act effectively as $c = \infty$ (Zheng and Kambhamettu, 2009).

5.3 Segmentation Results

Table 5.2 gives summary segmentation performance results for both ϕ_{all} and ϕ_{bb}, with Figure 5.5 displaying the distribution of scores across the VOT2016 dataset. The performance of simply predicting the entire bounding box to be the object, labelled as "Entire BB" in the figures, acts as a baseline and we note that on average the

Technique	Average performance	
	ϕ_{all}	ϕ_{bb}
Entire BB	0.579	0.747
OC-SVM + RGB	0.588	0.744
OC-SVM + LAB	0.581	0.745
OC-SVM + SIFT	0.579	0.747
OC-SVM + LBP	0.580	0.748
SBBM	0.555	0.747
Alpha Matting	0.763	0.804

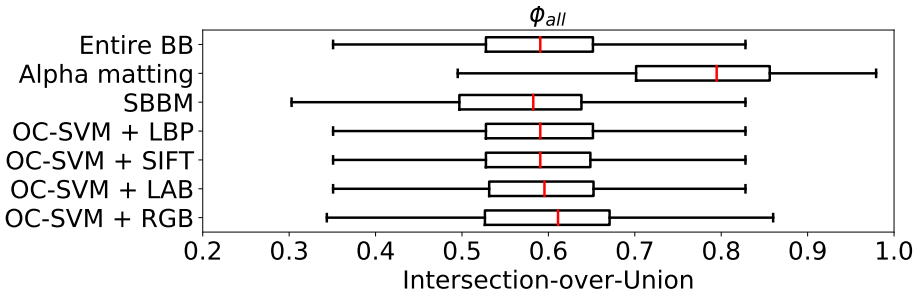
Table 5.2: Average segmentation performance for each of the techniques evaluated: the One-Class SVM (OC-SVM) alongside two colour (RGB and LAB) and two texture (SIFT and LBP) features, as well as Sample-Based Background Modelling (SBBM) and the alpha matting technique. These are compared to the baseline of predicting the entire bounding box as belonging to the object (Entire BB). The performance measures ϕ_{all} and ϕ_{bb} correspond to comparing the predicted segmentation to the ground-truth of the VOT2016 benchmark and comparing it to the ground-truth constrained to reside within the given bounding box. The first, second, and third best performances shown in their respective colours.

OC-SVM and SBBM methods perform similarly to the baseline. The range of scores is large for all methods (Figure 5.5), indicating that although each method performs well on some images, there others on which it performs poorly.

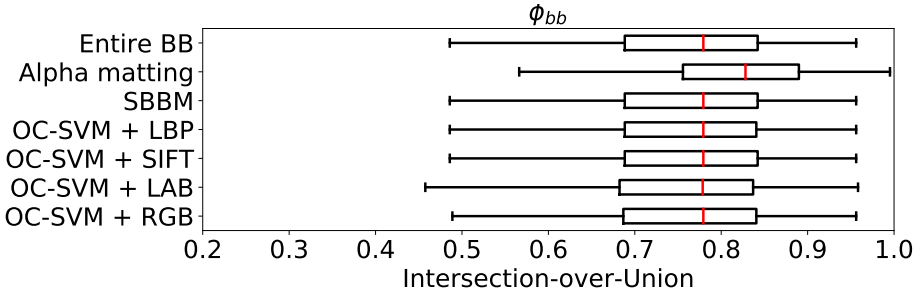
5.3.1 One-Class SVM

We compared segmentations using the cross-validated optimal parameters, with those using the parameters that gave the best result for ϕ_{all}. A typical example of this can be seen in Figure 5.6. In general, we found that the OC-SVM was capable of giving good segmentations, as can be seen in Figure 5.6c, but the parameters required to achieve these were widely spread, with no one region of parameter space giving good results for the majority of images.

It is interesting to note that there is little difference between the performance of the OC-SVM between the colour and texture-based features. In addition to the four feature-descriptors reported, we also combined them by concatenating the feature vectors together. No statistical improvement was found using any combination of features, and all suffered from the same problem as the four main features: good segmentations were too parameter-specific. Likewise, solely grey-scale features did not yield any improvement.



(a)



(b)

Figure 5.5: Performance characteristics of each technique for the two segmentation criteria, ϕ_{all} (a) and ϕ_{bb} (b). Each box plot shows the median (red), the first and third quartiles, as well as the minimum and maximum values on its whiskers.

These experiments were also carried out with an Isolation Forest (Liu, Ting and Zhou, 2008), a one-class classifier that can be used as an alternative to the popular OC-SVM. An exhaustive combination of features was evaluated, in a similar fashion to the OC-SVM. The classifier performed worse overall, achieving $\phi_{all} = 0.5174$ and $\phi_{bb} = 0.6330$ using RGB features, and scored comparably for other features.

5.3.2 Sample-Based Background Model

SBBM, like the OC-SVM, demonstrated the same problem of being too parameter dependent. We performed the same comparison of the cross-validated segmentations to the best possible segmentations, an example of which can be seen in Figure 5.7. The technique was capable of creating good segmentations on some of the test images, e.g. Figure 5.7c, but had a greater range of variation as to how successful the best possible segmentations were, similar to its interquartile range in Figure 5.5a.

5.3.3 Learning Based Digital Matting

The alpha matting technique achieved a much higher segmentation accuracy for both ϕ_{all} and ϕ_{bb}, as can be seen clearly in Figure 5.5, and improved on predicting the

(a) (b) (c)

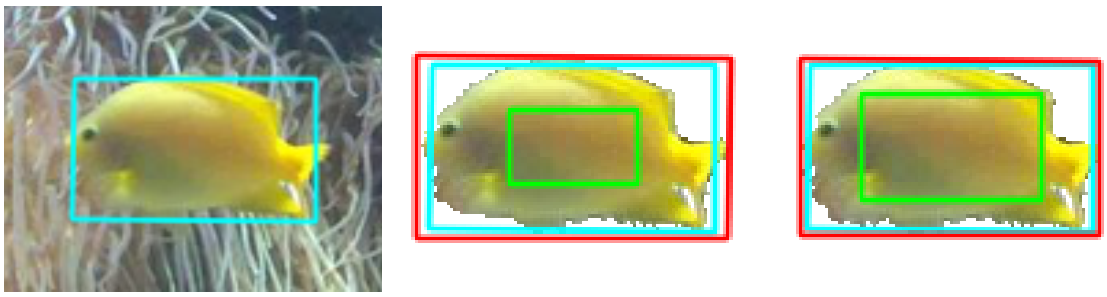
Figure 5.6: Example segmentations of the OC-SVM technique: Original image (a). Cross validated segmentation (b) ($\gamma = 10^6$, $\nu = 0.004$). Best possible segmentation using LAB features (c) ($\gamma = 10^{-19}$, $\nu = 0.250$). Note that the best possible segmentation has still misclassified a region of the athlete's back and foot.

(a) (b) (c)

Figure 5.7: Example segmentations of the SBBM technique: Original image (a). Cross validated segmentation (b) ($\delta = 0.1$, $\nu = 1.0$). Best possible segmentation (c) ($\delta = 0.3$, $\nu = 0.1$).

entire bounding box as the object by a considerable margin. It tended to perform very well in cases where the contracted bounding boxes, green in Figure 5.8, contained solely the object. However, when this region contained background pixels, as is the case in Figure 5.9b, these may be labelled as belonging to the object and propagated outwards. As Figure 5.9c shows, shrinking and translating the inner bounding box to include only the object improves the performance, but clearly this is not feasible in practice.

The group of objects whose centre of mass was positioned in some other location than the centre of the bounding box comprised almost completely of people. As people do not typically stand with their legs close together, and almost never do when in motion, we looked at the cross-validated performance of videos that contain humans (27) and those that do not (33). The average object segmentation



(a) (b) (c)

Figure 5.8: Example segmentations of the alpha matting technique: Original image (a). Cross validated (b) ($\rho^- = 0.8, \rho^+ = 1.2, \tau = 0.85, \lambda = 10^{-2}$) and best possible segmentations (c) ($\rho^- = 0.6, \rho^+ = 1.1, \tau = 0.85, \lambda = 10^{-2}$), with labelled original (cyan), expanded (red), and contracted (green) bounding boxes.

(a) (b) (c)

Figure 5.9: Using prior information to improve the segmentation performance of the alpha matting technique. Original image (a). Cross validated parameters (b) ($\rho^- = 0.8$, $\rho^+ = 1.2$, $\tau = 0.84$, and $\lambda = 10^{-2}$) and changing parameters to $\rho^- = 0.9$ and $\tau = 0.6$, and translating the inner bounding box up by 9 pixels (c).

performance ϕ_{all} for humans and non-humans was 0.718 and 0.800 respectively. Comparing this to the average performance across all videos (0.763) shows that segmenting humans is a harder than average task. This is in contrast to segmenting non-human objects, appearing to be a simpler task, as these typically are more compact. If one is able to detect the object to be tracked is human-like, using a combination of shrinking the inner bounding box further and translating it upwards, along with using a lower threshold τ for the amount of object likely in the bounding box, should improve performance somewhat.

Figure 5.10 shows histograms of the parameters which gave the highest performance for each tested frame. The extent to which the bounding box is shrunk to define the scribble mask is controlled by ρ^-. Figure 5.10a indicates that, for the

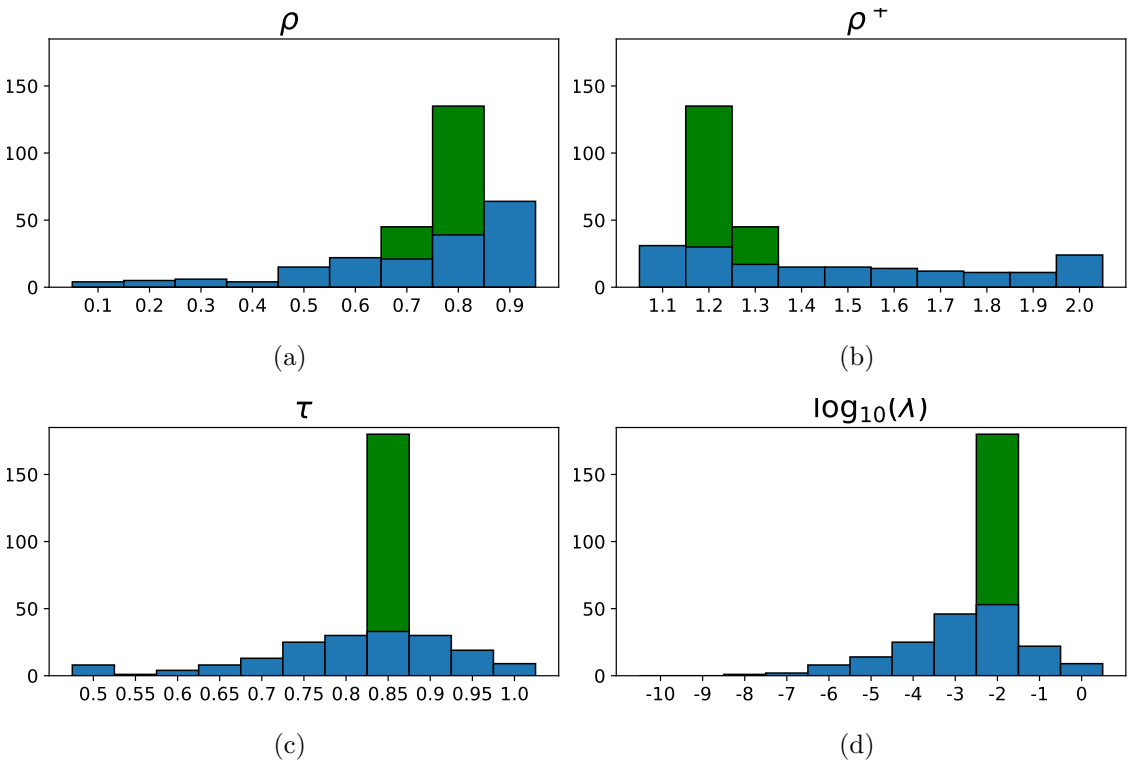


Figure 5.10: Distribution of the optimal (blue) and cross-validated (green) alpha matting parameters using the ϕ_{all} measure for the parameters controlling the bounding box contraction (a), the bounding box expansion (b), the alpha threshold (c), and the ridge regression shrinkage term (d).

majority of images, a large amount of shrinkage (small scribble mask) is optimal. This helps to avoid portions of the background for non-compact objects, such as humans. Figure 5.10b shows a wide variation in the optimal bounding box expansion parameter ρ^+, indicating that it sometimes helpful to include large parts of the adjoining regions, particularly for objects that extend outside the bounding box. However, the cross-validated optimum suggests an expansion of around 20%.

The values (Figure 5.10c) of the fraction of the bounding box that should be filled by the segmentation, τ, relate directly to the specific problem investigated. The benchmark (Kristan et al., 2016a) states that there should be no more than 30% of pixels in the bounding box containing background, which corresponds well to the region of most optimal parameters, 0.75 to 0.90 in Figure 5.10c. Figure 5.10d, showing the optimal value of the shrinkage term in the ridge regression, indicates a value between the two values recommended by the technique's authors ($\lambda = 10^{-1}$ in Zheng and Kambhamettu (2009) and $\lambda = 10^{-7}$ in their published code[1]). This may

[1] `http://www.mathworks.com/matlabcentral/fileexchange/31412`

indicate that the parameter is more problem-specific than they anticipated, and is possibly related to the size of the features used or complexity of the image.

The alpha matting experiments were also repeated with the neighbourhood $\mathcal{N}_i$ of each pixel defined as a 7×7 window centred on it. This resulted in approximately the same performance as using the 3×3 window, with $\phi_{all} = 0.760$ and $\phi_{bb} = 0.801$. Zheng and Kambhamettu (2009) recommended the use of a 7×7 neighbourhood size to give a more stable image matte, although their images, and therefore the number of unlabelled pixels, were much larger than here. The computational expense is dominated by solving Equation 5.12 for $\boldsymbol{\alpha}^*$. With 7×7 windows ($m = 48$) it is approximately forty times more expensive than for 3×3 windows ($m = 8$).

5.4 Conclusion

In this chapter, we propose three methods for determining which pixels in an image, for a given bounding box, belong to the object of interest and those that belong to its background. Two of these, using a One-Class SVM and our Sample-based Background Model, treat the problem as a one-class classification problem. They characterise the background of an image's colour/texture by modelling the features of superpixels that lie wholly outside the given bounding box. Features taken from the superpixels that lie inside or on the boundary of the bounding box are then compared to these models and are classified as belonging to the object if they are sufficiently different from the modelled features. Although both of these methods perform well with tuned parameters, neither are robust enough to allow for good performance on novel videos.

The third method uses an alpha matting method (LBDM) which seeks to separate an image into foreground and background components, based on user input. We automate the alpha matting process by automatically labelling pixels outside of an expanded version of the object's bounding box as background, and those within a contracted version of the bounding box as foreground. LBDM then learns the labels of the pixels located between the two labelled regions. This method gives good performance over the VOT2016 dataset using parameters chosen by cross-validation.

It is able to improve upon the assumption of most trackers that the entire bounding box belongs to the object. However, the method does rely on the assumption that the centre of the bounding box belongs to the object to be tracked and its performance degrades for non-compact objects for which this is not the case.

The LBDB method could be further adapted by using different features, such as those based on different colour models, e.g. HSV, LAB, or Color Names, (van de Weijer et al., 2009). Texture-based features, such as HOG, LBP, or SIFT may also give performance benefits, and could be combined with colour features. The method models a pixel's alpha matte value α_i as a linear combination of the alpha values of its neighbours (Equation 5.8). However, as the solution of this only uses the dot product of pixel features (Equation 5.16), it can be extended to a non-linear model via the use of the kernel trick (Zheng and Kambhamettu, 2009) and can therefore be used to represent more complex relationships between pixels can than be captured with a linear model.

There is further potential for performance increase by investigating what effect the value of c has on the segmentations. In the original LBDM problem formulation, its value was selected so that it could make maximal use of the user-provided pixel labels, as the assumption made was that the labels could be classed as always being the correct ones. However, in our case there is no guarantee that the automatically generated background and foreground labels are always correct, an so an investigation into allowing some deviation from the supplied labels may be able to take into account any erroneous labels provided.

In this chapter we have focused on the segmentation problem itself. However, the applications to the tracking problem are clear and, as discussed in Section 4.1, we use the alpha matting-based segmentation method as the basis of our patch placement scheme. In the following chapter, during the tracker component ablation study (Section 6.1), we compare the tracker's performance with and without the segmentation method. Since the publication of the work in this chapter (De Ath and Everson, 2018), Qin and Fan (2019) have investigated the use of our three object segmentation schemes to improve the performance of their Siamese Network-based

architecture. In addition to the template of the object fed into the network to search for in subsequent frames, they also include a template of the object having been segmented by one of our methods, thereby biasing the network to focus on the part of the bounding box that contains the object. They found that all three segmentation methods improved tracking performance in comparison to not performing any object segmentation, and found that the SBBM, alpha matting, and one-class SVM methods gave performance increases of 11%, 8%, and 6% respectively on the OTB benchmark (Wu, Lim and Yang, 2015).

All of these methods suffer from the paucity of data and we expect improved performance as higher resolution video becomes available. We note that the initialisation problem is significant not only at the start of tracking, but also when re-initialisation is required during tracking.

6. Empirical Evaluation

The performance of our tracker is now examined. In order to investigate the contribution of each main component of the tracker, we perform a component ablation study in Section 6.1 using the Visual Object Tracking 2018 benchmark (VOT2018). This is followed by a comparison of our tracking approach to part-based trackers submitted to VOT2018 as well as the top 3 performing trackers in the benchmark in Section 6.2. The tracker is also compared to the top-performing trackers on the Online Tracking Benchmark (OTB) in Section 6.3, and concluded in Section 6.5.

6.1 Ablation Study

In this section we perform a component ablation study of the tracker, in which we investigate the effect on performance of each key component in the final tracking method. The final tracker, named as Part-based Tracking by Sampling (PBTS) is comprised of the components with the parameter values that were selected and optimised in Chapters 3 and 4. These are detailed in Table 6.1, which shows each component of the tracker, its parameters, and their values, as well as the section in which they were evaluated. We first briefly recap the VOT2018 (Kristan et al., 2019) benchmark and then use it to compare the overall tracking performance of PBTS to five variants of it, each with a key component disabled or set to its default value. We then compare the performance of each tracker with respect to different visual attributes of the scenes being tracked.

The VOT2018 benchmark provides a supervised learning framework for evaluating short-term, model-free, causal trackers. As described in Chapter 2, it contains

Component	Parameter	Value
Initial bounding box segmentation (Section 5.3.3)	Bounding box contraction (ρ^-)	0.8
	Bounding box expansion (ρ^+)	1.2
	Proportion of bounding box filled (τ)	0.85
	Alpha matting regularisation (λ)	10^{-2}
Part placement (Section 4.1)	Patch location	Centroids
	Patch overlap amount (γ)	0.25
Part Model (Section 3.3)	Patch features	RGB
	Match radius (R)	20
	Number of patches (P)	35
	Patch side length (ω_x, ω_y)	5
	Maximum number of samples (S_{max})	10
	Modified Bhattacharyya distance (b)	1.4
Search scheme (Section 3.4)	Sets of patches to generate (G)	1000
	Sets of patches to locally optimise (L)	100
	Local optimisation window size (W)	5
	Sampling distribution (translation)	Laplace
	Sampling distribution (object size and rotation)	Gaussian
	Horizontal scale parameter (σ_x)	0.15
	Vertical scale parameter (σ_y)	0.1
	Object size scale parameter (σ_s)	0.02
	Rotation scale parameter (σ_r)	$\pi/16$
Model update (Section 4.3)	Features update rate (β_s)	1.7
	Counts update rate (β_c)	0.05
Part drift (Section 4.4)	Patch drift detection method	None
	Patch replacement method	None

Table 6.1: The optimal component attributes and their values for the final, optimised, tracker (PBTS), shown in their order of use and labelled with the section in which these values were evaluated.

60 specially selected videos, chosen such that they are both challenging and have a wide range of visual attributes that mirror real-world tracking scenarios. Each frame of video is manually annotated with a bounding box, along with one or more labels indicating which one of five visual attributes are affecting the object within the scene. These are either labelled as (i) *camera motion* (ii) *illumination change* (iii) *motion change* (iv) *occlusion* or (v) *size change*, or given a label of *empty*, indicating that these visual attributes are not present in the scene.

Trackers are evaluated 15 times on each video, and if their predicted bounding box has no overlap (IOU = 0, Equation 2.3) with the ground-truth bounding box then the tracker is marked as having failed and is reset 5 frames after the failure. Tracking performance is measured by three criteria: accuracy, robustness, and expected average overlap. Accuracy is given by the average overlap of the predicted

bounding boxes and the ground-truth bounding boxes, and the robustness is given by the failure rate (i.e. average number of failures per video). The expected average overlap (EAO) is an estimator of the expected average overlap the tracker would obtain given that it was evaluated on a set of videos with identical attributes to those in the benchmark. Further details of the evaluation protocol can be found in the benchmark protocol documentation (Kristan et al., 2019).

In the following, we vary one component of PBTS and keep all others fixed to the values of those in Table 6.1. The trackers to be evaluated are labelled:

- **PBTS**: The final, optimised, tracker with no changes.
- **No alpha matting**: Predicting the entire bounding box as belonging to the object during patch placement (Section 4.1). This increases the chances of background pixels being included in the part models.
- **Uniform patch placement**: Removing the entire patch placement scheme (Section 4.1) and placing patches uniformly over the given bounding box. This also increases the likelihood of background pixels being included in the part model.
- **Default MBD**: Using the standard definition ($b = \frac{1}{2}$) of the Bhattacharyya distance (Section 3.3). This relatively up-weights the contribution of poorly performing patches in the model, compared to the value used in PBTS ($b = 1.4$).
- **No local optimisation**: Performing no local optimisation (Section 3.4), meaning that the patches representing the object can only move in a non-shearing, affine, transformation.
- **No model update**: Turning off the model update component (Section 4.3) so that the initial samples and counts used to model the patch stay the same through the tracker's lifetime.

The overall performance results of these trackers can be seen in Table 6.2. Similarly to previous chapters, we note that the size of the standard deviations of the tracker's robustness is large but that their relative sizes between different trackers is still useful for comparison. The results show that each component of the tracker has a part to play in its overall performance, as demonstrated by the superior EAO of PBTS. Predicting the entire bounding box as belonging to the object (i.e. no alpha matting), in the same way of many other part-based trackers (see Section 2.1.4 for further details), is clearly detrimental to the tracking process as it results in a reduction in EAO of roughly 35%. It does, however, have the highest

Tracker	EAO	Average overlap	Robustness
PBTS	0.196	0.427 ± 0.101	1.723 ± 1.961
Default MBD	0.179	0.353 ± 0.104	1.658 ± 1.912
No local optimisation	0.169	0.396 ± 0.102	2.346 ± 2.218
No model update	0.162	0.386 ± 0.103	2.194 ± 2.421
No alpha matting	0.127	0.245 ± 0.105	1.500 ± 1.594
Uniform patch placement	0.118	0.234 ± 0.087	1.591 ± 1.563

Table 6.2: Results of the ablation study: comparing the PBTS tracker with individual components removed or set to their default value. The performance of the first, second, and third best trackers, for each respective attribute is shown in colour. The average overlap and robustness scores are averaged over each video.

robustness (lowest failure rate), which can be explained by the tracker predicting a larger region of the image as containing the object, which is also reflected in its lower average overlap score. Not using alpha matting in this case will mean that the entire bounding box is segmented into superpixels, and therefore will very likely contain superpixels that contain mostly (or completely) background pixels. Furthermore, as the tracker initialises its part model as the centre of superpixels, the tracker will have part models located at the centre of both foreground and background superpixels, resulting in the search scheme having the conflicting objectives of trying to both track the static background and the moving object.

Uniform patch placement, that is, placing patches uniformly across the bounding box, performs more poorly than placing them at the centres of superpixels, even if the entire bounding box is used as the object's location. This corresponds to the experiments in Section 4.1.2 that showed how tracking performance deteriorates when patches are initialised in inhomogeneous regions, as would be the case if placed uniformly across the bounding box. Removing model updating also led to a significant decrease in performance, as well as a significant increase in failure rate. Its average overlap score, however, is approximately the same as PBTS's, which, we suspect, is due to the method tracking the object well until the object's appearance sufficiently changes, resulting in tracking failure because the model is not updated to deal with the appearance changes.

Performing only a global, non-shearing, affine search for the object (i.e. not performing local optimisation on the patches) results in similar performance to not updating the model. The tracker has a slightly better average overlap than PBTS

and considerably worse robustness. This is an expected result because the tracker will only be able to generate non-shearing, affine transformations of the patches to evaluate patch positions in future frames. Consequentially, when the objects naturally deform in ways that cannot be described with a non-shearing affine transformation, tracking performance deteriorates rapidly, because the patches will no longer be able to match the object structure, and will, therefore, fail to accurately track the object.

Lastly, using the default Bhattacharyya distance parameter ($b = \frac{1}{2}$) in the modified Bhattacharyya distance (MBD, Equation 3.2) also gives a reduction in performance. This is a valuable result because it shows the importance of using a larger value of b (PBTS uses $b = 1.4$) to increase the difference between lower values of the Bhattacharyya coefficient (recall Figure 3.8). Increasing the distance between lower values of the Bhattacharyya coefficient (BC) corresponds to up-weighting the larger of two similar valued (in terms of BC) patches. This biases the model towards focusing more on the quality of the poorer performing patches and lessens the dominance of high BC-valued patches that will naturally occur when averaging a set of patch qualities.

Next we compare PBTS with the ablated trackers on the same benchmark with respect to each frame's visual attributes. The results of the attribute-based evaluation are summarised in Figure 6.1. Note that the robustness scores reflect the total number of failures per attribute and are not per video as in the previous evaluation. Similarly to Table 6.2, both the *uniform part placement* and *no alpha matting* trackers have consistently poor average overlap and high robustness. The latter again most likely due to the increasing size of its predicted bounding box as its patches drift. However, our placement strategy, even without segmenting the bounding box first, consistently outperforms uniformly placing patches across the bounding box. It is interesting to note that for all visual attributes the trackers generally follow a similar pattern in the graphs, indicating that they suffer from the same types of errors.

Both removing local optimisation and removing model update result in the same class of errors across all attributes, namely that they track the object well until

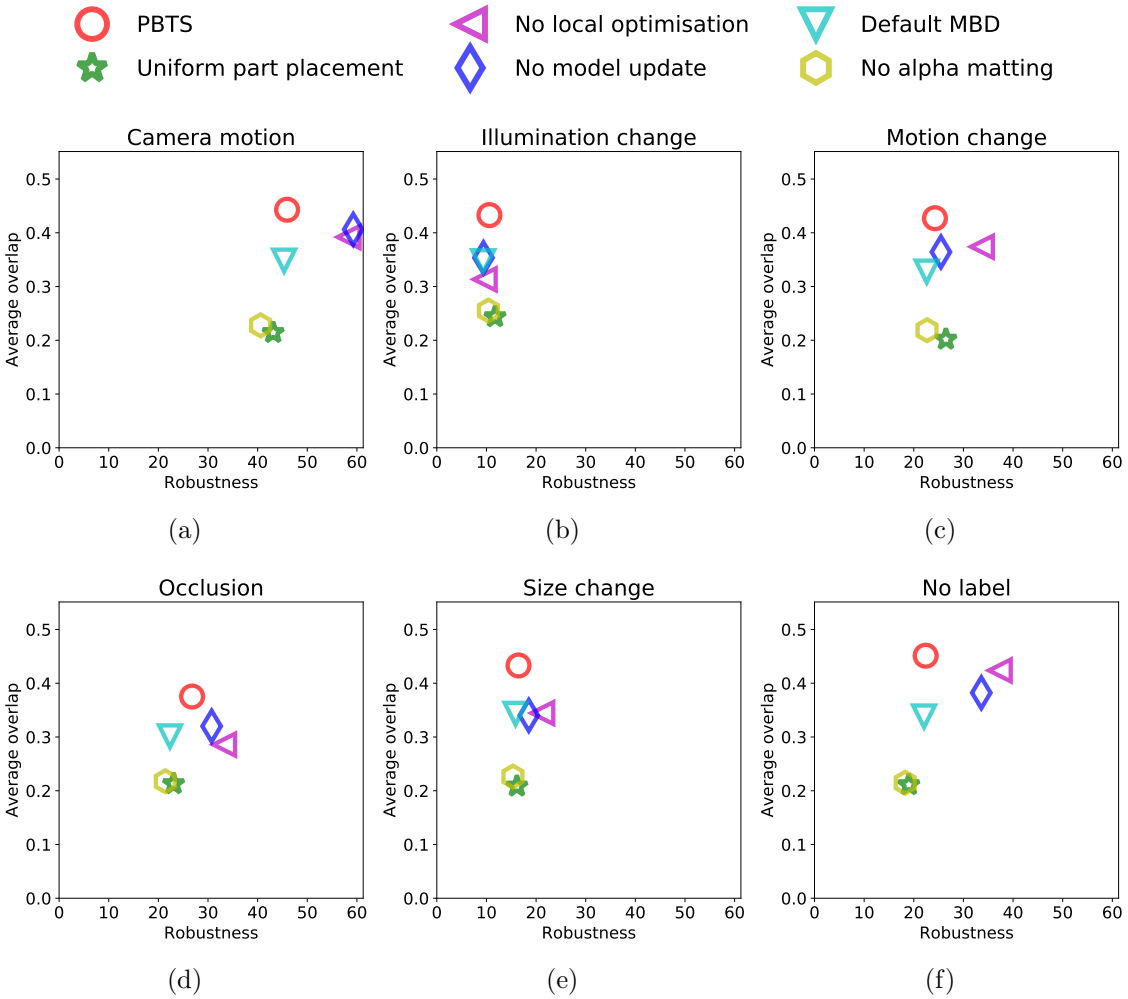


Figure 6.1: Performance of each tracker in the ablation study of PBTS (red) with respect to the visual attributes of each frame. The graphs show each tracker's performance in frames that contain *camera motion* (a), *illumination change* (b), *motion change* (c), *occlusion* (d), and *size change* (e). The final figure (f) shows the performance of the trackers on frames with no labelled attributes. Better trackers are located towards the top-left of each figure.

the object changes sufficiently (through deformation or appearance) and abruptly fail. This is reflected in their relatively high average overlap and low robustness (high rate of failure). Only changing the value of b results in performance similar to the full PBTS tracker, which is to be expected as all other components are activated. The figures indicate that increasing the value of b from the default Bhattacharyya distance value of $b = \frac{1}{2}$ leads a relatively large increase in accuracy at the cost of a slight decrease in robustness.

Illumination change (Figure 6.1b) appears to be the attribute that causes the most consistent failures for the tracker. This is reflected in the fact that the methods that are less robust in other attributes, such as performing no model update or local optimisation, have roughly the same failure rate with respect to illumination. Lack

of robustness to illumination is a common issue to colour-based trackers (Kristan et al., 2016a) because a rapid illumination change can cause colours to appear significantly different from how they looked (i.e. values in their respective feature space) in the previous frame. This suggests that colour features that specifically isolate illumination, such as LAB space (Section 2.1.1), would perform better in these situations. However, as the experiments in Section 3.3.2 show, we found little evidence that this is the case when using LAB or HSV features in our tracker.

This ablation study shows the importance of each component in the tracker. Placing patches in regions of the bounding box that are more likely to contain the object, i.e. using our patch placement methodology, significantly increases the accuracy of tracking, as is shown by the poor robustness by placing patches uniformly or not including the bounding box segmentation. While it would be surprising if this was not the case, it is still nevertheless an important result because it demonstrates that this is a fruitful line of investigation for future work. As discussed in Section 2.1.4, researchers have previous appeared to given scant consideration towards the placement of object parts and therefore we hope this work will encourage others to actively pursue tracker initialisation.

Modelling object deformation by locally optimising patch locations provides a substantial increase in robustness, allowing the object parts to better follow the changing geometric appearance of the tracked object. The tracker will be further analysed with respect to the changing geometry of objects (deformation) on the OTB benchmark in Section 6.3 as the benchmark provides labels for videos based on deformation. Updating the object model also yields a substantial improvement to the robustness of the tracker, which is also to be expected because the object part models will be able better adapt to the object's changing appearance. Section 4.3.2 provides more details the model update scheme and shows the performance of disabling either part of the update model (i.e. setting $\beta_s = 0$ or $\beta_c = 0$).

Now that the importance of each component of the tracker as been established, we evaluate PBTS (Table 6.1) on the VOT2018 benchmark and compare it to other part-based trackers as well as the top 3 performing trackers on the benchmark.

6.2 Comparison to Other Trackers on VOT

In this section, we give context to the tracker's performance compared to other part-based methods by evaluating PBTS using the VOT2018 benchmark. We compare the tracker to the other 10 part-based trackers submitted to the benchmark, as well as the top 3 performing trackers. We note here that the benchmark (Kristan et al., 2019) contains the result of an earlier version of the PBTS tracker, labelled as PBTS-A (Table 3.3) in this thesis, and was used as a baseline for the earlier experiments.

The ten part-based trackers evaluated on the VOT2018 benchmark were ANT (Čehovin, Leonardis and Kristan, 2016a), DPT (Lukežič, Zajc and Kristan, 2018), LGT (Čehovin, Kristan and Leonardis, 2013), DFPReco[1], FoT (Vojíř and Matas, 2014), BST (Battistone, Petrosino and Santopietro, 2017), BDF (Maresca and Petrosino, 2015), Matflow (a combination of BDF and Matrioska), FragTrack (Adam, Rivlin and Shimshoni, 2006), and Matrioska (Maresca and Petrosino, 2013). The top 4 part-based trackers are all similarly structured and use a global model to guide the parts when tracking the object. LGT uses 3 components for its global model, capturing the colour, motion and shape of the object, and greyscale histograms for its object parts. ANT, which is based on LGT, uses a kernalised correlation filter (KCF, Henriques et al., 2015) for its global model and the same part model. DPT uses both a KCF and colour histograms as its global model and uses KCFs for its parts. Similarly, DFPReco uses a more advanced correlation filter, ECO (Danelljan et al., 2017), as both its global and local models.

The top 3 performing trackers on the benchmark are LADCF (Xu et al., 2018), MFT (Bai et al., 2018), and SiamRPN (Li et al., 2018). Both LADCF and MFT are use a correlation filter-based formulation, with LADCF using adaptive spatial regularisation to train a low-dimensional correlation filter. MDF combines the output of multiple correlation filters trained on features extracted at different image resolutions, similar to image pyramids. SiamRPN combines a Siamese network, used

[1] DPRFeco was submitted to the VOT2018 benchmark without any published work associated with it. The authors are A. Memarmoghadam and P. Moallem {a.memarmoghadam,p_moallem}@eng.ui.ac.ir.

Tracker	EAO	Average overlap	Robustness
PBTS	0.196	0.427 ± 0.101	1.723 ± 1.861
ANT	0.168	0.439 ± 0.140	2.250 ± 2.494
DPT	0.158	0.463 ± 0.142	2.567 ± 3.499
LGT	0.144	0.403 ± 0.103	2.641 ± 2.542
DFPReco	0.138	0.467 ± 0.155	2.983 ± 3.879
FoT	0.130	0.385 ± 0.157	3.667 ± 2.791
BST	0.116	0.244 ± 0.106	3.136 ± 3.065
BDF	0.093	0.336 ± 0.131	4.200 ± 4.389
Matflow	0.092	0.362 ± 0.132	4.550 ± 4.380
FragTrack	0.068	0.325 ± 0.139	6.650 ± 6.483
Matrioska	0.065	0.365 ± 0.154	6.900 ± 8.451
LADCF	0.389	0.507 ± 0.133	0.567 ± 0.803
MFT	0.385	0.496 ± 0.153	0.500 ± 0.847
SiamRPN	0.383	0.567 ± 0.119	0.983 ± 1.072

Table 6.3: Results of PBTS on the VOT2018 benchmark compared to 10 other part-based trackers (upper section of the table) and 3 state-of-the-art trackers (lower section). The performance of the first, second, and third best performing part-based tracker with respect to each performance metric is shown in colour. Note that the average overlap and robustness scores are averaged over each video, and so are per video scores. The results for all other trackers are extracted from Kristan et al. (2019).

for feature extraction, with a region proposal network to predict object locations. Note that we include the results of these three trackers to illustrate the difference in performance between part-based methods and the current state-of-the-art methods.

The EAO, average overlap, and robustness of the trackers on the VOT2018 benchmark is shown in Table 6.3. As can be seen from the table, PBTS outperforms all other part-based trackers on the benchmark in terms of both EAO and robustness, and places fourth in terms of accuracy. Given the relative simplicity of the tracker, compared to those using global models to guide tracking on top of their local models, our part-based methodology performs well. However, compared to the current state-of-the-art methods (those in the lower portion of Table 6.3) our tracker performs considerably poorer with respect to all three attributes. We suggest that this is due to both the tracker architecture used and the types of features used for object representation. The ten highest performing trackers on the benchmark all use deep neural network features (combined with various other hand-crafted features) in either a correlation filter-based or CNN-based framework (Kristan et al., 2019).

Next we compare PBTS to the other trackers with respect to each frame's visual attributes. The results of this can be seen in Figure 6.2. We compare

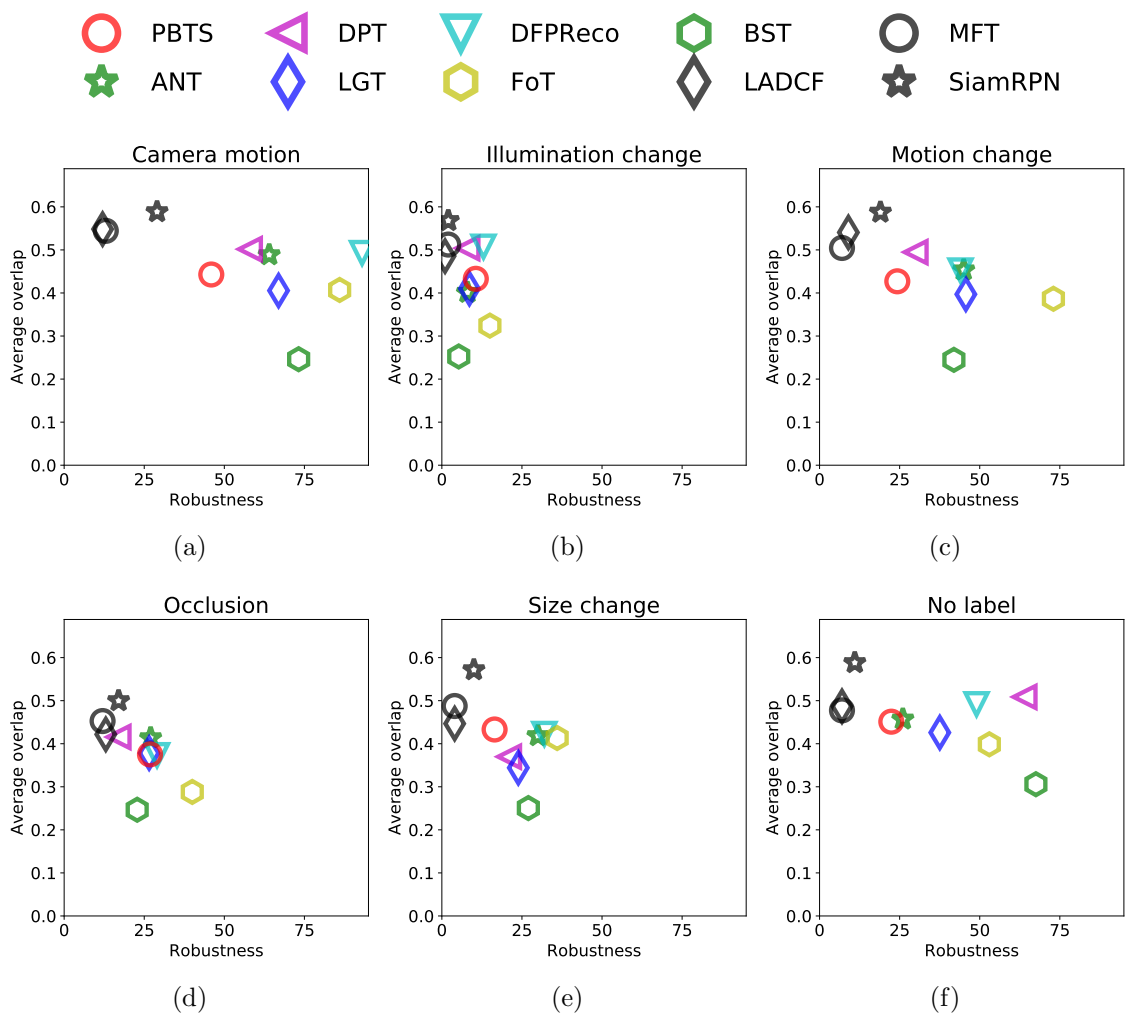


Figure 6.2: Performance of PBTS (red) on the VOT2018 benchmark with respect to the visual attributes of each frame, compared to the other part-based trackers (coloured) submitted to the benchmark and also the top 3 performing trackers (black). The graphs show each tracker's performance in frames that contain *camera motion* (a), *illumination change* (b), *motion change* (c), *occlusion* (d), and *size change* (e). The final figure (f) shows the performance of the trackers on frames with no difficult attributes. Better trackers are located towards the top-left of each figure. Note that Matrioska, FragTrack, Matflow and BDF have been left off the plots for clarity due to their relatively low performance.

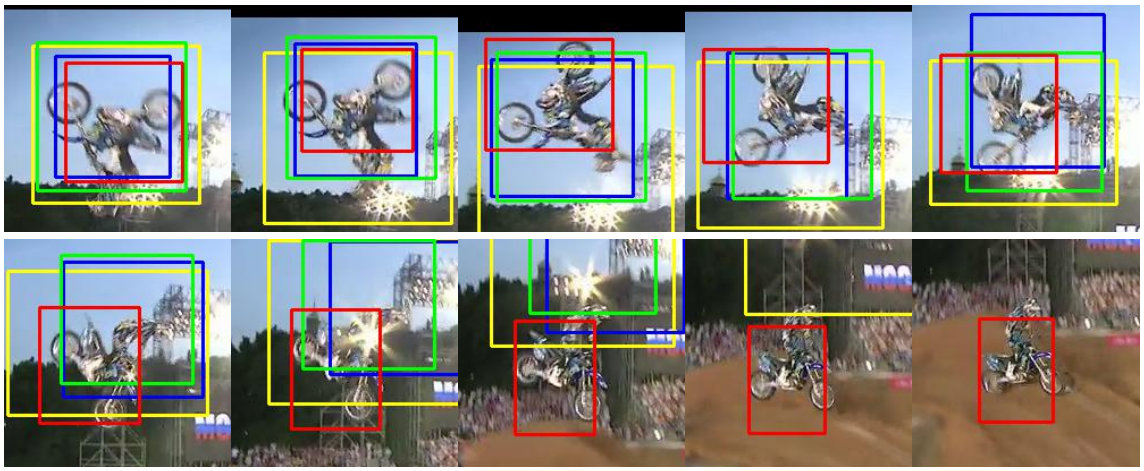
the results of PBTS (red) to the other part-based trackers (coloured) and leave the state-of-the-art trackers (black) in the plots as a guide to what is the highest performance possible with respect to each attribute. We do not report the results of BDF, Matflow, FragTrack, and Matrioska due to their poor performance. PBTS performs comparatively well with respect to object motion in terms of *camera motion* (Figure 6.2a) and the object itself changing motion (Figure 6.2c). We suggest that this is because they are similar attributes in the sense of how they can be modelled, because they can both lead to large jumps in the object's location. PBTS deals with this well because, when generating sets of candidate part locations, it uses

Latin Hypercube Sampling to generate points in object translation space to sample from the Laplace distribution (Section 3.4.6) that allows for large translations to be modelled at each iteration. This gives the tracker the ability to reach objects that have made large movements (in pixels) between consecutive frames.

Part-based approaches appear to fail roughly equally due to *illumination changes* (Figure 6.2b), however both the correlation filter-based methods, DPT and DPRFeco, are more accurate than PBTS. This may be explained by the types of features that they use to represent object parts, because both trackers use HOG features (Dalal and Triggs, 2005). These are somewhat invariant to illumination and shadowing (Dalal and Triggs, 2005), and therefore may explain the tracker's increased accuracy because they would be better able to track object parts that are rapidly changing appearance due to illumination. LGT and ANT both use greyscale histograms for image parts and therefore are far more susceptible to rapid changes in illumination, similar to the RGB features of PBTS.

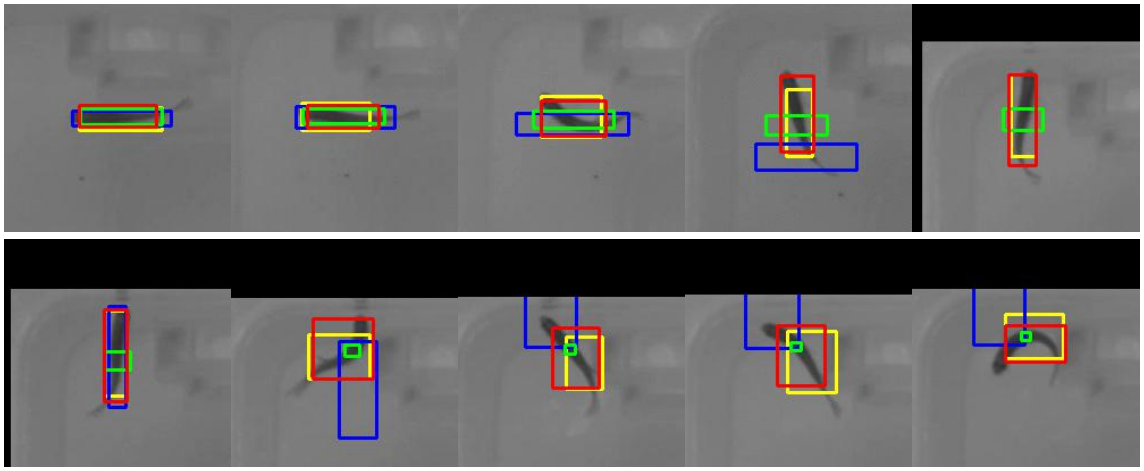
Occlusion (Figure 6.2d) appears to be a similar problem for all part-based approaches to some extent, however, DPT performs better in terms of robustness to the other methods. This is because the tracker explicitly takes into account occlusion in two ways: firstly, it has quite a rigid structure between parts, meaning that it is hard for parts to drift away from one another. Secondly, only updates its object part models when they have a quality above a certain threshold, which allows one or more parts to be completely occluded with no loss of tracking accuracy if the other object parts are tracking well. PBTS outperforms all other part-based trackers with respect to frames containing significant object *size change* (Figure 6.2e). We suspect that this is because our tracker can sample from a wide range of potential scales (Section 3.4.6) and can, therefore, better capture extreme changes in object size.

Qualitative tracking results for PBTS compared to ANT, DPT, and LGT are shown in Figure 6.3. In the `motocross1` sequence (Figure 6.3a), PBTS is the only tracker of the four to be able to cope with the object rotating in and out of plane, as well as large flashes of light from the floodlights (right of images). We suspect that this is partly due to the enhanced patch placement procedure that PBTS carries out.



(a) `motocross1`.

(b) `ants1`.



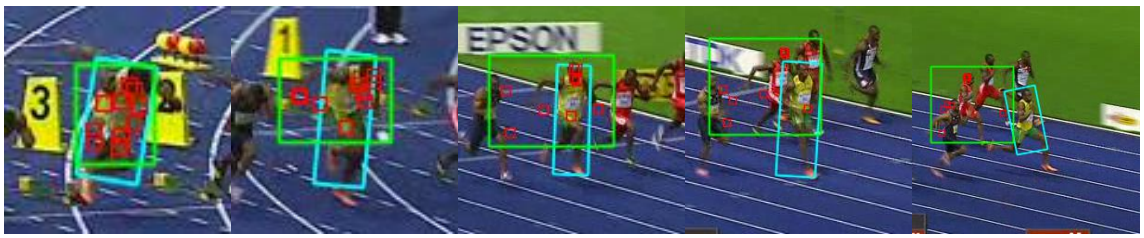
(c) `zebrafish1`.

Figure 6.3: Qualitative tracking results on several challenging sequences from the VOT2018 benchmark for PBTS (red) compared to three best other part-based trackers, ANT (green), DPT (blue), and LGT (yellow). Each set of images are sampled from the first 40 frames in the `motocross1` (a), `ants1` (b), and `zebrafish1` (c) sequences. Bounding boxes are omitted for frames that a tracker has failed on. Black borders of some frames are shown in order to maintain the aspect ratio at the edge of the cropped images.

Since the bounding boxes will naturally contain lots of background information (e.g. sky), placing object parts arbitrarily across the bounding box will likely cause many parts to be tracking the background, which is avoided by the PBTS patch placement procedure (Section 4.1). This effect is reflected in the large expansions of the other tracker's bounding boxes over the first few frames of video. We note, however, that by the end of the sequence of images PBTS has started to drift, as patches have drifted away from the lower part of the motocross rider.

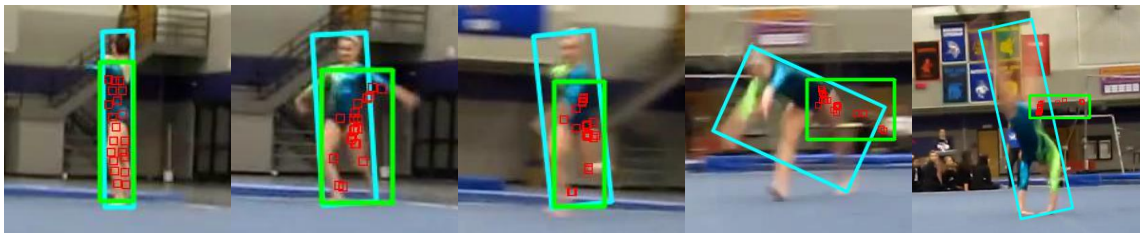
Tracking results for the `ants1` sequence (Figure 6.3b) show PBTS coping with object rotation well, whereas both ANT and DPT fail to track the ant as it quickly rotates 90 degrees. LGT also tracks the ant without failure but starts to drift as the ant rotates, most likely due to the tracker initialising its parts on the edge of the bounding box in the first frame. Similarly to the motocross sequence, the ant sequence also contains a non-convex object which means that a higher proportion of background will be included in trackers that arbitrarily place patches. Lastly, the `zebrafish` sequence (Figure 6.3c) shows PBTS also tracking the object well. The DPT tracker, which splits the object's bounding box into four equal parts such that each part takes up one quadrant of the object. This results, in this case, in very long and thin parts being generated that quickly fail to match the fish as it deforms (i.e. bends as it rotates). ANT also performs poorly as the object rotates, but manages to continue to track a small region of the object. Both PBTS and LGT continue to track the object throughout the sequence, with PBTS providing a better estimate of the object's true location.

Three examples of failure cases of the PBTS tracker are illustrated in Figure 6.4. In the `bolt1` video (Figure 6.4a), some patches are initialised partially on the background of the runner due to a poor segmentation. This leads to those patches tracking the background, while other patches, initialised on the runner's torso when it was in shade, match poorly as he moves into the light and they too start to drift. Eventually the majority of patches coalesce on the runners head and jump to another runner's head due to its similar colour, leading to failure. The tracker does not fail completely due to any one visual attribute, but slowly fails over time due to the



(a) `bolt1`.

(b) `rabbit`.

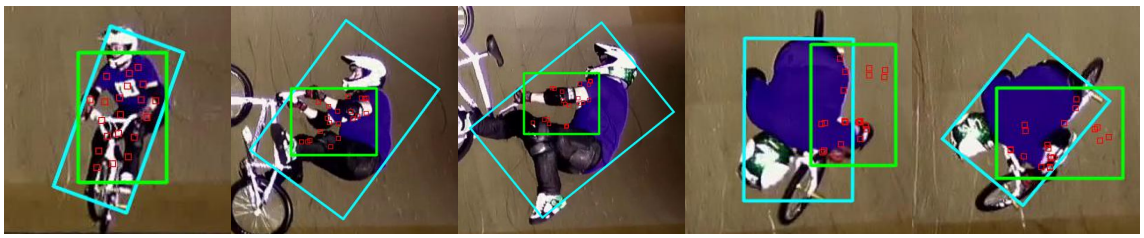


(c) `gymnastics2`.

Figure 6.4: Examples of PBTS failing to track objects in the challenging sequences `bolt1` (a), `rabbit` (b), and `gymnastics2` (c), taken from the VOT2018 benchmark. The tracked object parts are shown in red, with the predicted and ground-truth bounding boxes shown in green and cyan respectively. Note that the right-most frame of each sequence is the last frame before the tracker predicts a bounding box that has no overlap (IOU = 0, Equation 2.3) with the ground-truth bounding box.

challenging nature of the scene resulting from shadows, self occlusion by the runner's arm, and motion blur.

In the `rabbit` sequence (Figure 6.4b), the rabbit is very similar in appearance to its background, leading to patches drifting off the rabbit as it moves across the snow. PBTS relies solely on a colour model, and therefore struggles to track object that are very similar to their background, due to them being potentially indistinguishable in colour space. In the last sequence, `gymastics2` (Figure 6.4c), the gymnast rotates out of plane, undergoes large amounts of deformation, moves at a high speed (relative to the frame-rate of the camera recording), and is filmed by hand-held, moving, camera. These challenging visual attributes combined to rapidly change the object's appearance, resulting in the tracker underestimating the scale of the object and eventually losing track of the object completely.



(a) `bmx`.

(b) `basketball`.

(c) `fish2`.

Figure 6.5: Examples of PBTS robustly tracking objects but incorrectly estimating their bounding boxes on the `bmx`, `basketball`, and `fish2` sequences of the VOT2018 dataset. Tracked object parts are shown in red, with the predicted and ground-truth bounding boxes in green and cyan respectively.

Lastly, we explore several instances where PBTS robustly tracks an object (i.e. without failure), but has a poor reported accuracy (overlap with the ground-truth bounding box). This is illustrated in Figure 6.5. The `bmx` (Figure 6.5a) and `basketball` (Figure 6.5b) sequences show instances in which several patches have drifted away from the object. This has resulted in a poor prediction of the object's bounding box because, even though the majority of patches are correctly adhering to the object, the drifting patches have caused either an under- or over-estimation of the region. Naturally, one remedy would be to employ drift detection, but as we have shown in Section 4.4, this is not a trivial problem to solve and, although the schemes we propose are helpful in some circumstances, on average they decreased performance.

In the `bmx` sequence (Figure 6.5a) in particular, the patches that are tracking the background are very close to those that are tracking the BMX rider, meaning

that trying to identify them as drifting patches based on their location would not be feasible. Similarly in the `basketball` sequence, the player moves out his left leg and the patches correctly track it, but then as the player moves his leg back in the patches latch onto the dark markings on the basketball court which are similar in colour the player's socks. In this case, we would not want the position of the patches after they have followed the leg outwards to be classed as drifting, but after the player moves their leg back in we would want to label them as drifting. This is a difficult scenario as neither the position of the patches, nor their match quality (as they match well to the markings on the court) is indicative of whether the patch is an outlier.

The `fish2` sequence (Figure 6.5c), highlights another source of reduced overlap. In this case an axis-aligned bounding box does not appropriately describe the shape of the object, because the fish is facing diagonally downwards. Given that it would be possible to create a mask of the best PBTS estimate of the object location by taking the either the visual or convex hull of the patches, a rotated minimum area rectangle could be fitted to the mask. Another potential solution to improve accuracy in these situations would be to use the automatic bounding box generation method proposed in the VOT2016 benchmark (Kristan et al., 2016a). These two methods, as well as our strategy of fitting an axis-aligned bounding box to predicted object mask, have been recently explored by Wang et al. (2018). They found that using the VOT benchmark's method gave the best accuracy on the VOT2018 benchmark. We note that, regardless of the method used to generate the bounding box of the object, the real accuracy of the tracker remains unchanged. However, in future work it would be interesting to analyse the general performance of part-based trackers with respect to the different bounding box prediction strategies available.

6.3 Comparison to Other Trackers on OTB

The OTB benchmark contains additional sequences to the VOT20018 benchmark that were published in recent years (Wu, Lim and Yang, 2015), and can be viewed as a complementary evaluation to that of VOT. This is because, in contrast to the

VOT benchmark, its main challenge, the one-pass evaluation (OPE), focuses on unsupervised tracking, i.e. a tracker is initialised only once per sequence. This does mean, however, that trackers which explicitly include re-detection mechanisms will obviously be at an advantage when compared to the those trackers evaluated on the VOT benchmark, which generally do not include any form of long- term tracking components.

We evaluate PBTS on the OTB100 benchmark (Wu, Lim and Yang, 2015) and compare its performance to the top-performing trackers. Given that PBTS uses only colour information during tracking (RGB features), we evaluate it on the 75 colour sequences of the OTB100 benchmark. To ensure a fair comparison, the performance of PBTS will be compared with other trackers using their published results on the same 75 colour videos. The OTB benchmark defines 11 visual attributes to label tracking challenges, these are: (i) *background clutter*, (ii) *deformation*, (iii) *fast motion*, (iv) *in-plane rotation*, (v) *illumination variation*, (vi) *low resolution*, (vii) *motion blur*, (viii) *occlusion*, (iv) *out-of-plane rotation*, (v) if the target moves *out-of-view*, and (vi) *scale variation*. In contrast to the VOT benchmark, OTB only provides per sequence attributes, which means that the reported performance of a tracker on a certain attribute in the benchmark will actually only be given for a subset of frames included in the performance measure.

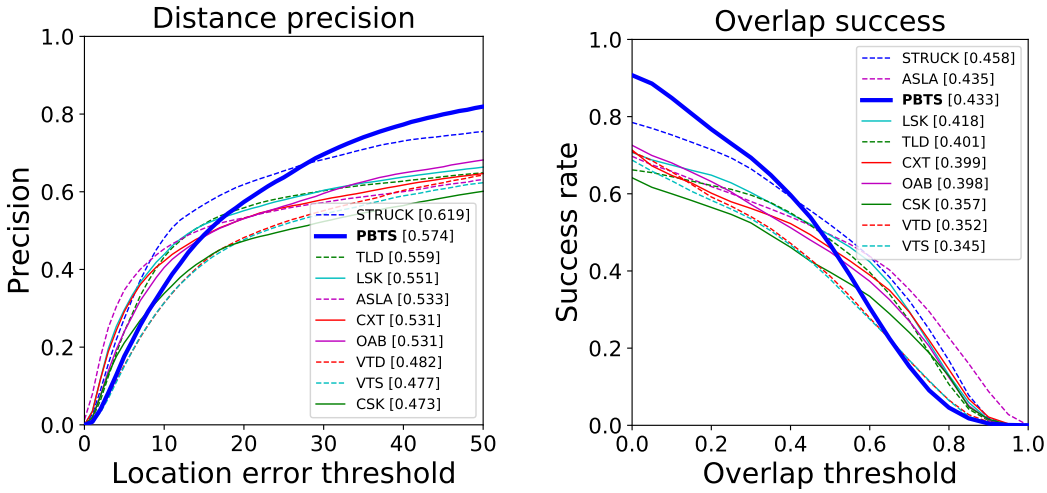
The OTB benchmark measures performance using success and precision plots. A success plot is a measure-threshold plot of the proportion of tracked frames that have an IOU (Equation 2.3) larger than a threshold, which is varied from 0 to 1. It also has an associated summary statistic used to quickly compare trackers, which is the area under the success plot. A precision plot similar to the success plot, is a measure-threshold plot that shows the proportion of frames that are a centre error (Equation 2.2) less than a threshold, ranging from 0 to 50 pixels. Its summary statistic is the proportion of frames for which the centre error is less than 20 pixels.

We compare PBTS to the top-performing trackers on the OBT-100 OPE benchmark, namely STRUCK (Hare et al., 2016), CXT (Dinh, Vo and Medioni, 2011), ASLA (Jia, Lu and Yang, 2012), TLD (Kalal, Mikolajczyk and Matas, 2012),

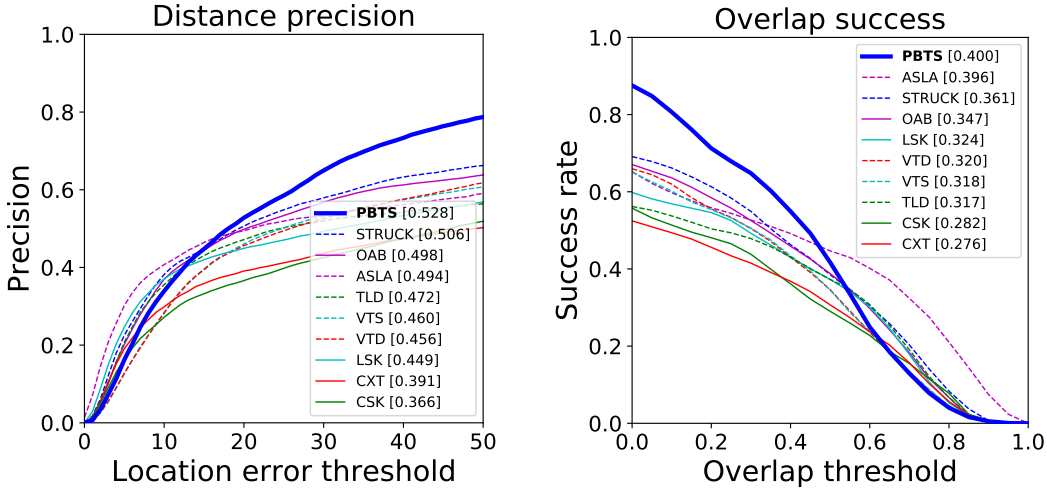
CSK (Henriques et al., 2012), LSK (Liu et al., 2013), VTD (Kwon and Lee, 2010), OAB (Grabner, Grabner and Bischof, 2006), and VTS (Kwon and Lee, 2011). STRUCK is tracking-by-detection algorithm that uses an online structured output Support Vector Machine. CXT is a distractor-aware algorithm that monitors regions of the image that are similar to the tracked object and down-weights their locations when matching their object model. ASLA use a structural locally sparse appearance model to perform incremental subspace learning. TLD is a long-term tracking framework that decomposes the long-term tracking problem into three parts, tracking an object between frames, localising the object's appearance and correcting the tracker, and learning the related error rates of the components so that they can be updated if needed.

CSK is a correlation-filter based approach that trains a correlation filter using greyscale features in the Fourier domain via kernel ridge regression. LSK uses selection-based dictionary learning algorithm with a locally constrained, sparse, object representation. VTD is based on a visual tracking decomposition scheme that has multiple object models, constructed by sparse principal component analysis, and combines each model's object location estimates with an interactive Markov Chain Monte Carlo (MCMC) framework. OAB tracks objects uses online AdaBoost feature-selection to select the most discriminating features in order to successfully differentiate the tracked object from its background. VTS uses multiple trackers to estimate the location of a tracked object, as well adding and replacing poorly performing trackers as it detects tracking a change in conditions by sampling new trackers from a predefined tracker state space using MCMC.

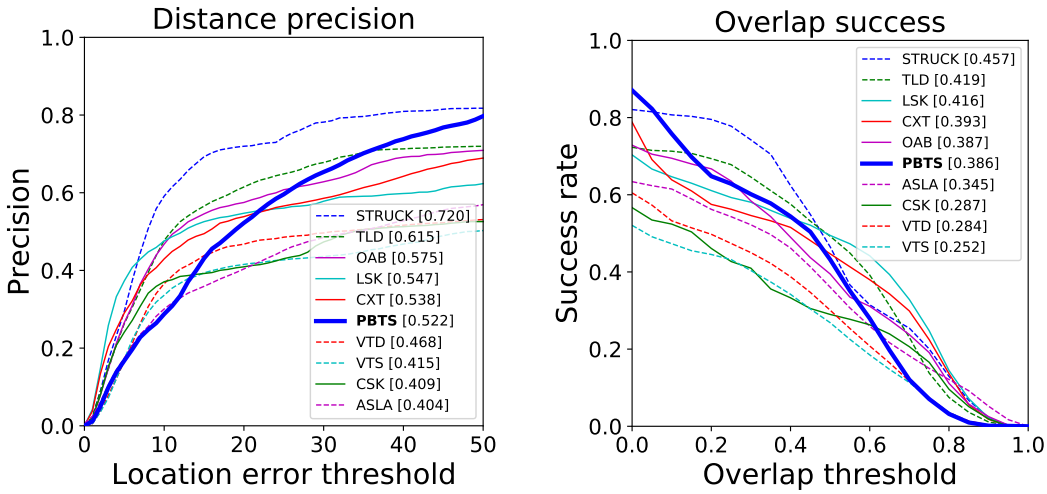
The performance of PBTS over all colour sequences is shown in Figure 6.6a, with the per attribute performance shown in Figures 6.6–6.9. Overall, PBTS performs well on the benchmark, achieving second and third place in terms of precision and overlap respectively. The general shape of the success plots of PBTS tends to be more S-shaped than the majority (e.g. Figure 6.6a), which we suspect is due to the underestimation of the bounding box previously discussed in Section 6.2. PBTS generally achieves a higher level of precision than the other trackers once the location



(a) Results over all 75 colour sequences.

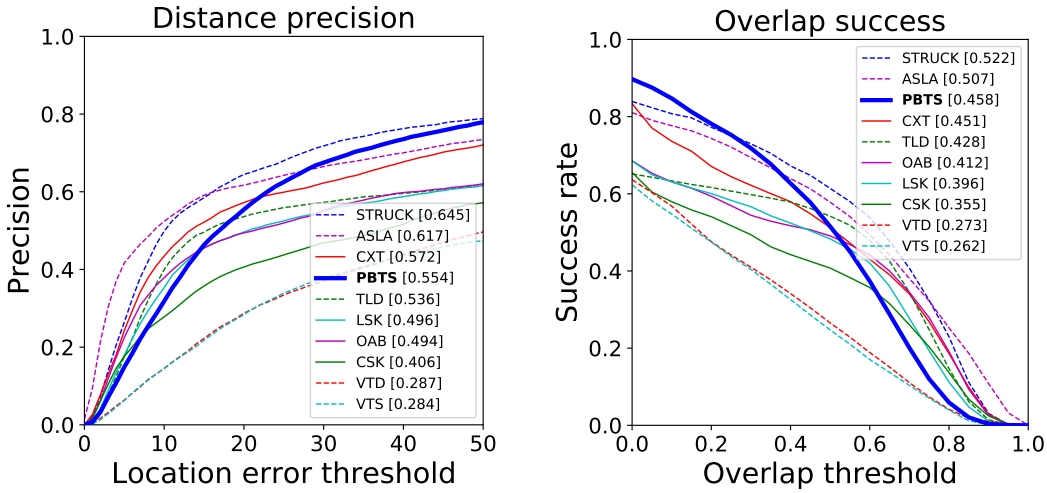


(b) Results for the 37 sequences with the attribute *deformation*.

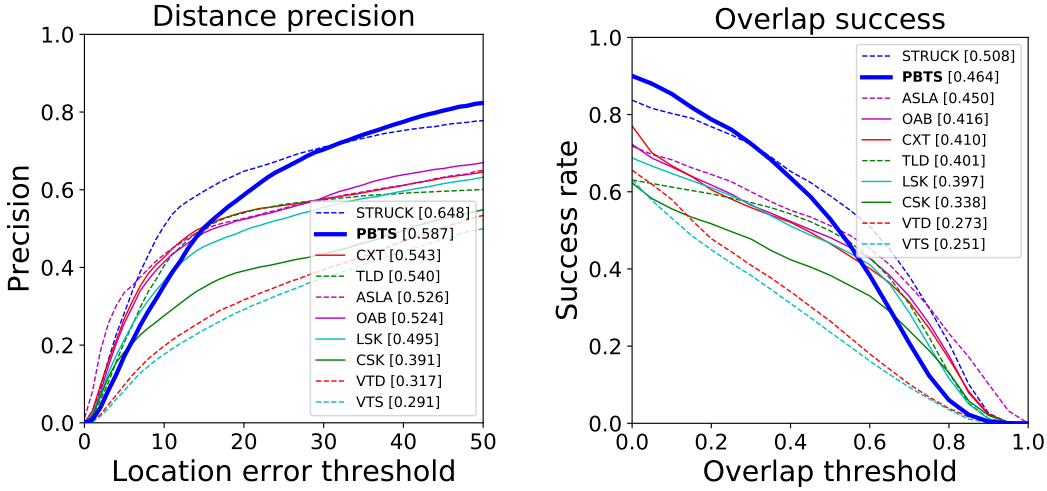


(c) Results for the 9 sequences with the attribute *low-resolution*.

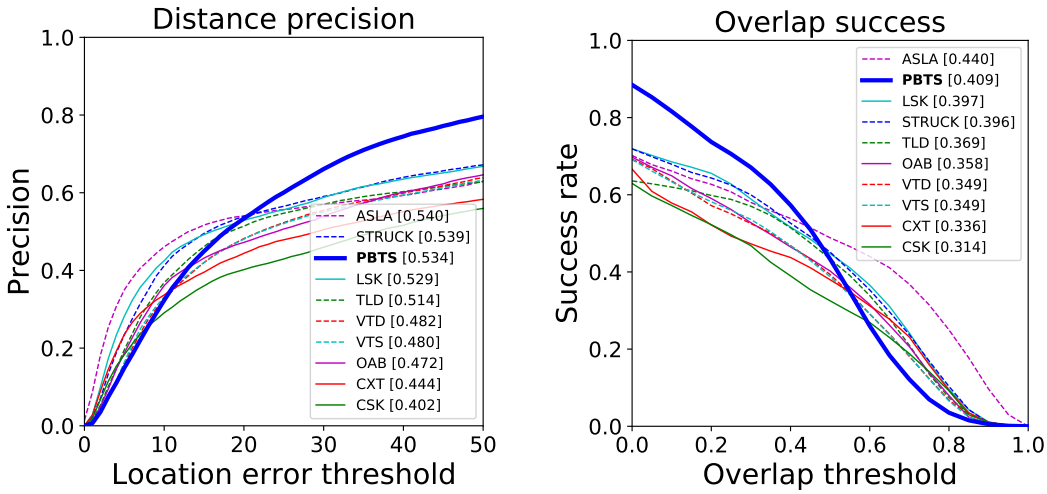
Figure 6.6: Performance results on the OTB100 dataset for all colour sequences (a), as well as the attributes *deformation* (b) and *low resolution* (c). The left hand figures show the distance precision (centre distance) plots and the right hand figures show the success (IOU with respect to ground-truth) plots. Each plot's legend orders the trackers by the value of the associated statistic, the percentage of frames with a centre distance less than 20 pixels (left) and area under the success curve (right). Better trackers have distance precision curves towards the top-left corner of the plots and overlap success curves towards the top-right corner of the plots.



(a) Results for the 26 sequences with the attribute *motion blur*.

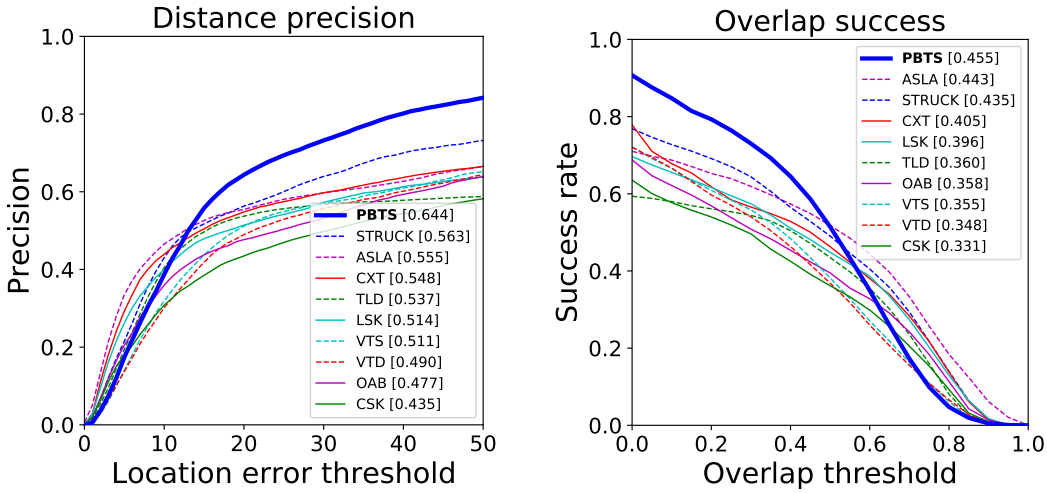


(b) Results for the 34 sequences with the attribute *fast motion*.

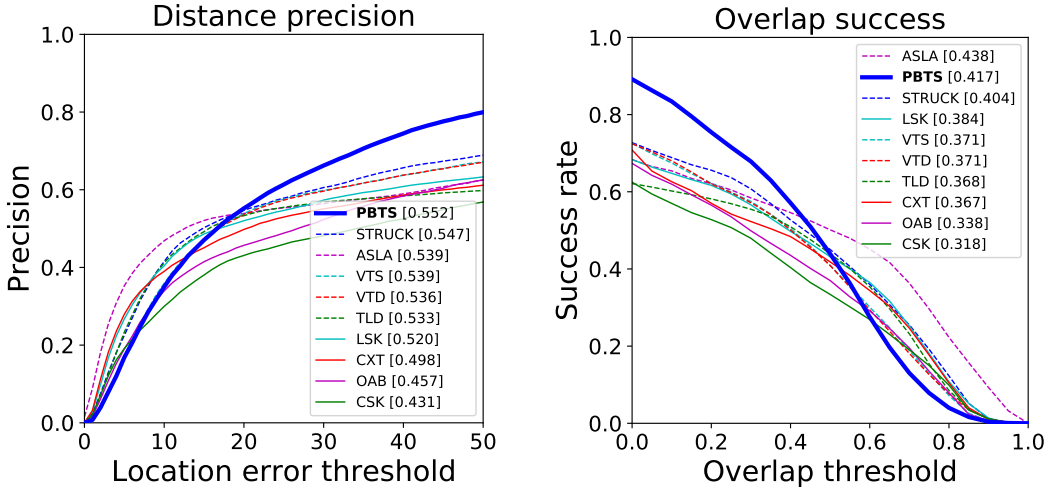


(c) Results for the 42 sequences with the attribute *occlusion*.

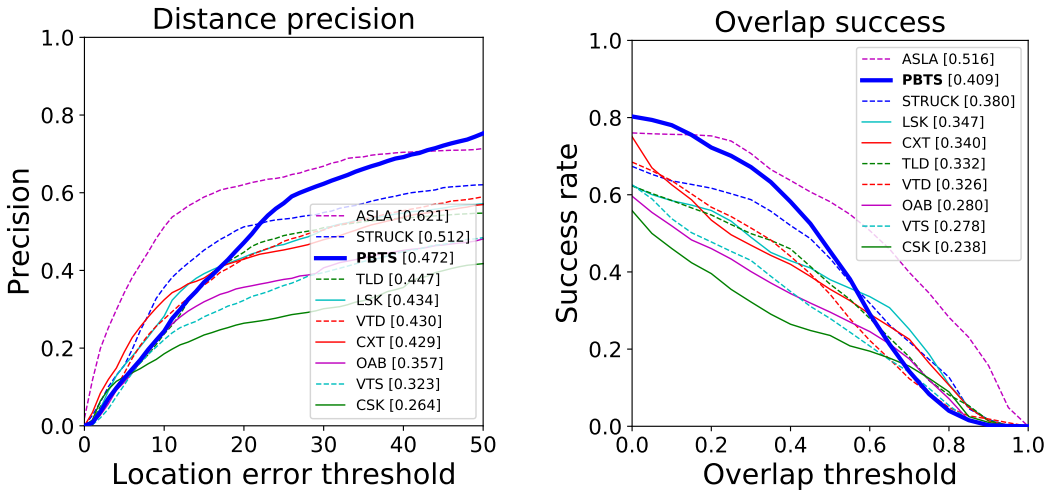
Figure 6.7: Performance results on the OTB100 dataset for the attributes *motion blur* (a), *fast motion* (b) and *occlusion* (c). The left hand figures show the distance precision (centre distance) plots and the right hand figures show the success (IOU with respect to ground-truth) plots. Each plot's legend orders the trackers by the value of the associated statistic, the percentage of frames with a centre distance less than 20 pixels (left) and area under the success curve (right). Better trackers have distance precision curves towards the top-left corner of the plots and overlap success curves towards the top-right corner of the plots.



(a) Results for the 34 sequences with the attribute *in-plane rotation*.

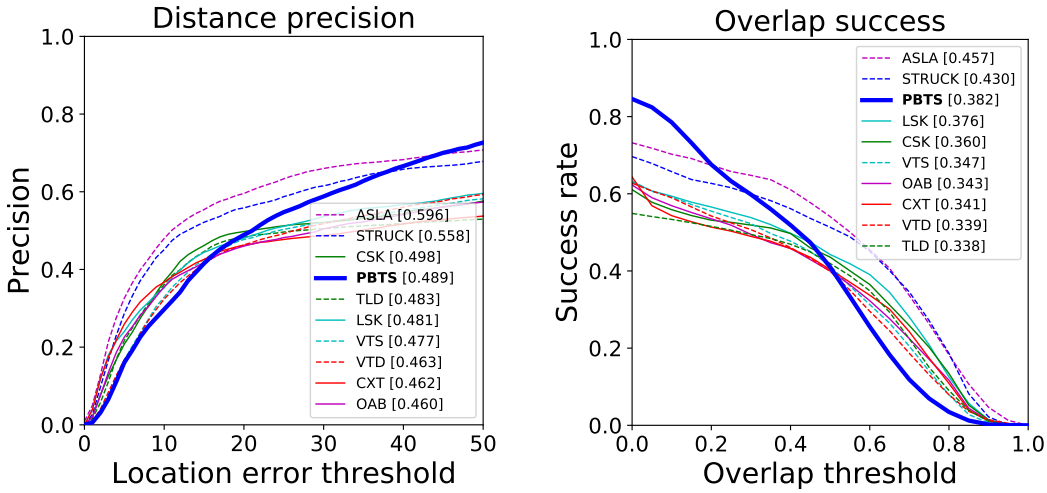


(b) Results for the 49 sequences with the attribute *out-of-plane rotation*.

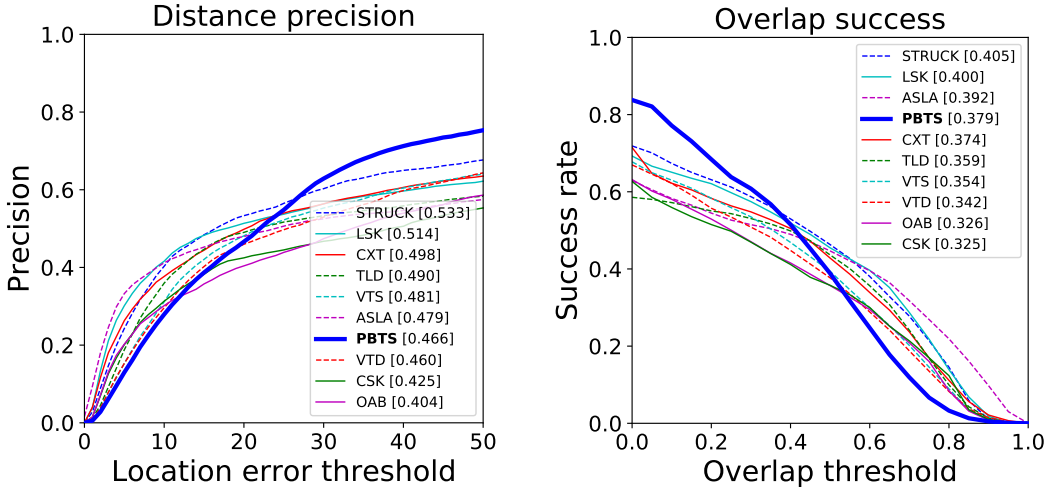


(c) Results for the 11 sequences with the attribute *out-of-view*.

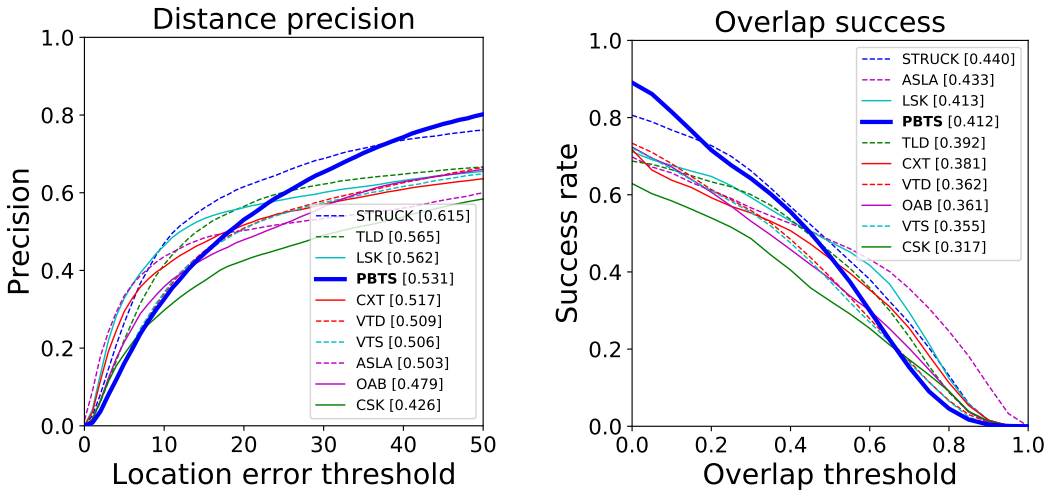
Figure 6.8: Performance results on the OTB100 dataset for the attributes *in-plane rotation*(a), *out-of-plane rotation* (b) and *out-of-view* (c). The left hand figures show the distance precision (centre distance) plots and the right hand figures show the success (IOU with respect to ground-truth) plots. Each plot's legend orders the trackers by the value of the associated statistic, the percentage of frames with a centre distance less than 20 pixels (left) and area under the success curve (right). Better trackers have distance precision curves towards the top-left corner of the plots and overlap success curves towards the top-right corner of the plots.



(a) Results for the 25 sequences with the attribute *background clutter*.



(b) Results for the 31 sequences with the attribute *illumination variance*.



(c) Results for the 50 sequences with the attribute *scale variation*.

Figure 6.9: Performance results on the OTB100 dataset for the attributes *background clutter* (a), *illumination variance* (b) and *scale variation* (c). The left hand figures show the distance precision (centre distance) plots and the right hand figures show the success (IOU with respect to ground-truth) plots. Each plot's legend orders the trackers by the value of the associated statistic, the percentage of frames with a centre distance less than 20 pixels (left) and area under the success curve (right). Better trackers have distance precision curves towards the top-left corner of the plots and overlap success curves towards the top-right corner of the plots.

error threshold is around (or above) 25 pixels. We believe that this is due to the robustness of the part-based formulation of the tracker, as demonstrated by its performance on the VOT2018 benchmark. If several object parts have drifted, the majority will continue to track the object, despite the fact that the drifted parts may be a significant distance from the main group of parts tracking the object (recall Figure 6.5b). This means that the predicted bounding box's centre may be away from the centre of the object, but closer than typical holistic trackers (which model the entire bounding box using one appearance model). In the holistic case, if the tracker starts to model a region containing a large amount of background, the entire tracker will tend to drift off the object, resulting in failure.

PBTS performed the best on videos that contained *deformation* (Figure 6.6b), *in-plane rotation* (Figure 6.8a), and *out-of-plane rotation* (Figure 6.8b). This is to be expected because the search process (Section 3.4.1) explicitly models in-plane rotation via sampling non-shearing affine transformations. The search processes also models object deformation via the part local optimisation scheme (Algorithm 3, Section 3.4.1), and given that out-of-plane rotation can be viewed as an extreme type of deformation, it can also handle this well. The tracker also performs well at tracking objects that are moving *fast* (Figure 6.7b), which confirms the performance of the tracker on the VOT2018 benchmark (Figure 6.2c). Somewhat surprisingly, given that the tracker does not have any in-built occlusion handling, it achieves third place with respect to the attributes *occlusion* and *out-of-view*[2]. This is, however, similar to the tracker's performance on the VOT2018 benchmark (Figure 6.2d) in which the tracker performed roughly as well as the other trackers in the benchmark.

PBTS ranked fourth in tracking performance with respect to object *scale variation* (Figure 6.9c), but we do note that there is very little difference between the top 4 tracker's AUC scores. The lower performance with respect to scale variation may be somewhat deceptive, as it represents an average over 50 of the 75 videos, and therefore actually contains a vast number of frames of videos that do not contain scale variation, bringing the performance much closer to the overall performance of

[2] Note that an object being *out-of-view* refers to it being out of the bounds of the image, which, we would argue, is equivalent to *occlusion* but in a different location of the image

each tracker (Figure 6.6a). We suspect that if the frames were individually labelled as to when the object changed scale, the tracker would achieve a higher measured performance, similar to its ranking in the VOT2018 benchmark with respect to size change (Figure 6.2e). The tracker only achieves fourth in videos with the attributes *motion blur* (Figure 6.7a) and *background clutter* (Figure 6.9a). Given that motion blur can cause the object's colour appearance to change markedly, we consider this to be reasonable performance. This mirrors the performance of PBTS on the VOT2018 benchmark, where it performs the most robustly (out of part-based trackers), but has a lower accuracy, with respect to both motion change and camera motion.

The attribute *background clutter*, referring to other objects and background in a tracking video that are visually similar to the tracked object, was also handled surprisingly well by PBTS. Given that PBTS has no in-built notion of its surroundings, unlike the CXT tracker that performed substantially worse in this respect, we view this as an encouraging result. However, the drift detection schemes explored in Section 4.4, suggested that the tracker did have patches that would drift off to distracting regions of its surroundings. Therefore, given that we know PBTS is somewhat susceptible to *background clutter*, that its relative placement only indicates that the other, equally performing, trackers also find the clutter difficult to robustly deal with.

The performance of PBTS with respect to *illumination variation* is somewhat lower than the other trackers, ranking 7th out of the top 10 trackers on the OTB100 benchmark. Relying solely on a colour-based model naturally leads to problems with illumination, as is also reflected in the VOT2018 results (Figure 6.2b). As previously stated, this is because the colour of an object is strongly related to the its current illumination, meaning that if large changes in lighting occur then the colours of the object will significantly change. This is a difficult problem for all of the evaluated trackers, as shown by the relatively similar performance of trackers on both the VOT and OTB benchmarks. However, the state-of-the-art, deep learning-based methods in the VOT2018 benchmark do perform slightly better than the trackers using more traditional, hand-crafted, features in the OTB one. Therefore one potential avenue

for future work would be to look at integrating deep features into the tracker, as this has been shown to be successful in correlation filter-based approaches (Danelljan et al., 2015a).

Lastly, PBTS also performs poorly (ranked 6th out of the top 10) on the attribute *low resolution*, which indicates that the object's ground-truth bounding box has an area of less than 400 pixels in the first frame of a sequence. We suspect that this is due to a combination of several factors. An area of roughly 400 pixels corresponds to a region of size 20×20 pixels, meaning that only a small number of patches of 5×5 pixels can be placed onto the original object. In the *ideal* situation, where the entire bounding box is predicted to belong to the object and the region is superpixeled into square regions of size 4×4 pixels, it would still not be possible to fit 35 patches (5×5 pixels) onto the object because the patches would overlap by more than 25% ($\tau = 0.25$). The largest number of patches placed in this scenario would be 13 placed in a chequerboard pattern. However, this scenario is unlikely as some of the bounding box will be labelled as background during its initial segmentation (Section 5.3.3). Consequentially, in future work, it would be worth exploring whether a greater degree of patch overlap or using smaller patches is helpful to combat problems encountered when tracking small objects.

6.4 Computational Complexity

In this section we will briefly outline the computational cost of each part of the tracking pipeline and evaluate the processing time of each frame of the VOT2018 benchmark. The computational cost of the tracker can be split into two main parts, the initialisation procedure, which is dependent on the size of the object, and the main tracking pipeline, which is object size independent.

The initialisation process starts by cropping a frame, centred on the object, to twice the size of the object's width and height, and performing the alpha matting procedure, which itself consists of two computationally expensive processes. Firstly, the calculation of the local linear alpha colour model coefficients (Equation 5.16) is carried out for each pixel within the cropped region around the given bounding box.

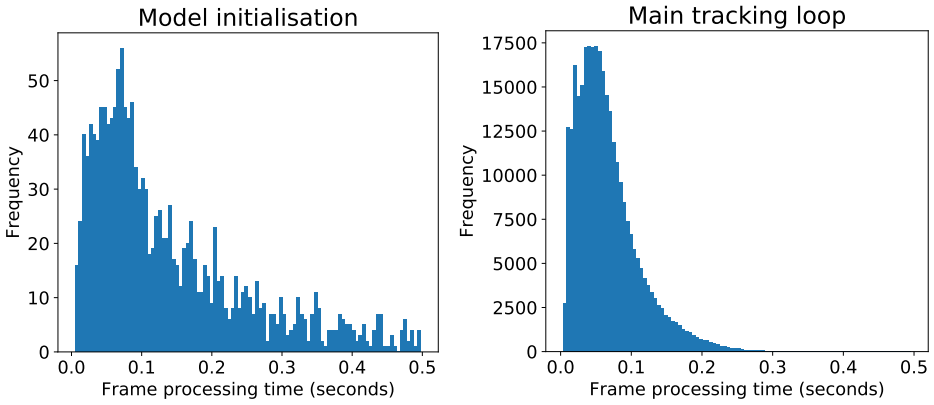
Secondly, a linear system of equations (Equation 5.12) is solved in order to calculate the alpha matte values for each pixel. This gives the process $\mathcal{O}(m^3)$ complexity, where m is the number of unlabelled pixels within the cropped region.

The cropped image region is then superpixeled using the SLIC algorithm, which has $\mathcal{O}(N)$ complexity (Achanta et al., 2012), where N is the number of pixels in the cropped region. Lastly, P patch models are created, centred on the largest superpixels, such that the superpixels reside within the region indicated by the alpha matting procedure as likely to contain the object. This is carried out by Algorithm 5, and, for each placed patch, carries out $|\Omega_p|(|\Omega_p|+1)/2$ Euclidean distance calculations in RGB feature space, where $\Omega_p \in I$ is the patch region.

Once the tracker has been initialised, the main tracking loop is independent of the size of the object. It, instead, depends on the number P and size $|\Omega_p|$ of patches, as well as the number G of candidate sets of patch locations to evaluate, the number L of these to locally optimise, and the size W of the local optimisation window. The evaluation of a candidate patch's quality (Algorithm 1) carries out, $|\Omega_p| \cdot |\mathcal{M}_p|$ distance calculations in RGB feature space, where $|\mathcal{M}_p|$, the number of feature-count pairs in a patch's model, grows sub-linearly with time as more features are added and removed from the patch's model over the length of the video. This evaluation is carried out $G + LW^2$ times for each patch in the model.

The tracker was implemented in Python, with the patch quality evaluation procedure (Algorithm 1) written in cython[3], a C-like extension to the Python programming language. During the previous sets of experiments in this chapter, the tracker ran on a machine with an Intel Xeon E5-2690 v4 CPU and 512GB RAM. Figure 6.10 shows the distribution of the frame processing time for all frames processed in the evaluation of the VOT2018 benchmark corresponding to the results in Table 6.3. Figure 6.10a shows the time taken to (re)initialise the tracker across the evaluation runs, and Figure 6.10b shows the time taken to perform the main tracking loop (i.e. all operations after initialisation has taken place). The benchmark records the processing time of each frame evaluated, and includes the time taken to read the frame from disk. We note that the times towards the right-hand side of the

[3] `www.cython.org`



(a) (b)

Figure 6.10: The distribution of time taken for PBTS to process a frame of video from the VOT2018 dataset, for both frame initialisation (a) and all subsequent frames (b) of a video sequence. Note that these computation times include the time taken to read in an image from a mechanical hard drive.

Figure 6.10a correspond to larger objects, and those towards the right hand side of Figure 6.10b correspond to larger images.

These per-frame times correspond to a mean performance of roughly 7 frames per second for the initialisation process and 15 frames per second for the main tracking loop, carried out once the tracker has been initialised. This is slower than some of the fastest trackers, which are typically based on Siamese Networks (e.g. Li et al., 2018) or correlation filters (e.g. Xu et al., 2018), and achieve frame rates in excess of 50 frames per second. We do, however, note that the naive implementation of PBTS used in this work is limited to running one core of a CPU, and therefore could be sped up significantly by calculating the patch quality evaluations, which are independent of one another, in parallel across multiple cores of the processor. There is also potential for speed increases in the tracker initialisation process, as an obvious extension to the algorithm would be to pre-scale all images to ensure their given bounding boxes were the same number of pixels in area. This would result in the same computational performance for all images, aside from the additional overhead of resizing the images, and is a common preprocessing step in trackers using as correlation filters (e.g. Xu et al., 2018) and Convolutional Neural Networks (e.g. Nam and Han, 2016).

6.5 Conclusion

In this chapter we investigated the performance of the tracker's main components in an ablation study on the VOT2018 benchmark (Section 6.1), compared its overall performance to other part-based and state-of-the-art trackers on both the VOT2018 (Section 6.2) and OTB100 (Section 6.3) benchmarks, and investigated its computational complexity (Section 6.4). Throughout each analysis we compared the tracker's performance with respect to the specific visual attributes labelled for videos in each benchmark.

The ablation study compared the final version of the tracker, PBTS, which consisted of the components and associated parameter values determined throughout Chapters 3-5 and summarised previously in Table 6.1. It was found that each of the core components, initial patch location selection, using a modified Bhattacharyya distance, local optimisation, and model update all played an important role in the tracker's overall performance. The most important component, indicated by the largest performance loss when it was removed, was that of the patch placement scheme. Placing patches uniformly across the bounding box of an object led to an EAO reduction of approximately 40%, with a similar performance loss (35%) also given by not segmenting the object within the given bounding box. We suspect that the performance of other, part-based, tracking methods could be improved significantly by implementing this type of initial step to identify regions of the initial bounding box that are more likely belong to the object.

The second main finding of the ablation study that is applicable to other part-based trackers is difference in performance of the tracker when using the default ($b = \frac{1}{2}$) version of Bhattacharyya distance (MBD, Equation 3.2) and our modified version ($b = 1.4$). This increases the distance between lower values of the Bhattacharyya coefficient (BC, Equation 3.1) more than the distance between higher values of the BC. This biases the model towards selecting a set of candidate patches that has a higher quality of its lowest performing patches, decreasing the potential dominance of high BC-valued patches. Using the MDB may give a noticeable increase in performance for other, part-based, trackers that use the Bhattacharyya distance,

such as ANT (Čehovin, Leonardis and Kristan, 2016a) and BHMC (Kwon and Lee, 2009), and those that collectively evaluate a set of object parts, such as WPCL (Zhu et al., 2015), DGT (Cai et al., 2014), and GGTv2 (Du et al., 2017).

PBTS was also compared to other part-based techniques on the VOT2018 benchmark (Section 6.2), as well as to the top 3 performing trackers on the benchmark (Kristan et al., 2019). PBTS performed better than the next best part-based tracker (ANT) in terms of EAO by 15%, as well has being significantly more robust but with a slightly lower average overlap. The tracker performs worse than the state-of-the-art methods with respect to all three performance measures. However, we note that PBTS uses far simpler features (RGB) than any of the top-performing trackers on the benchmark. One natural piece of future work would be the exploration of using deep neural network features or texture descriptors to improve part matching.

Problems in tracking for PBTS were also analysed. Tracking failure mainly occurred when the background of the object became sufficiently similar to the tracked object, resulting in patches slowly drifting off to track the background. Failures also occurred when a series of challenging circumstances occurred at the same time, such as self occlusion, illumination changes, and motion blur. The former failure is conceptually easier to address because the tracker needs to be aware of its surroundings, i.e. being *distractor-aware* as in the work of Zhu et al. (2018). This could be accomplished by modelling the tracker's background and down-weighting any visual matches that occur there. The latter problem, however, is harder to address as the tracker does not fail from one type of visual attribute but many that combine to completely change the appearance of the object from that which was modelled. Again, the use of different features may make the differences between extreme changes object appearance less severe and lead to better tracking performance.

We also compared the tracker to the top-performing trackers on the OTB100 benchmark using the one-pass evaluation methodology (i.e. no reset upon tracking failure). PBTS achieved second and third place in terms of precision and overlap respectively. In both evaluations on the OTB and VOT benchmarks it was noted that the tracker achieved lower levels of overlap than other trackers, while mostly being

more robust. In the OTB benchmark in particular, the S-like shape the tracker's success curves indicate that it is does not usually estimate the object's bounding box perfectly but still continues to track the object well. The poor overlap success rate results from several patches drifting from the object while the rest track it successfully, resulting in a larger bounding box prediction than the ground-truth's. The issue of drift was discussed in Section 4.5, in which several suggestions for future pieces of work in this area were considered.

In both the OTB and VOT benchmarks we compared PBTS to the other trackers with respects to the labelled visual attributes of the datasets. PBTS performed well with respects to attributes that it explicitly modelled, such as general object motion, object size change, and in-plane and out-of-plane rotations. All of these visual attributes are modelled accurately by the search scheme (Section 3.4) because it samples possible translations, rotations, and scale changes from a probability distribution, rather than looking within a fixed range. As a result of this, the tracker is able to model larger translations, for example, better than other techniques that assume an object can only move within a fixed window, such as correlation filters (Henriques et al., 2015). The tracker also out-performed all other trackers on the OTB benchmark when tracking objects that undergo substantial deformation.

Conversely, the tracker performed more poorly with respect to attributes that it does not model, such as large illumination changes, background clutter, and small objects. As a part-based method, tracking smaller objects can prove challenging because fewer parts can be fitted into a small bounding box, and, as discussed in the previous section, can lead to can increase in background being included into the patch models. We suggest a possible remedy for this is to adapt the patch placement scheme to perform differently when objects are sufficiently small and fewer than the desired number of patches can be fitted. In particular, decreasing the number of patches to be placed, increasing their size, and allowing for more patch overlap should allow the object to be sufficiently covered without placing sections of patches into the known background region.

Illumination changes and occlusion, both by other objects in the scene and the

object moving out of view, also proved a problem for PBTS. Illumination changes are difficult for colour-only trackers to address, as changes in illumination cause the object's colour to change, resulting in a model mismatch. We note that, as previously mentioned, additional features would be needed in order to improve the tracker's robustness to illumination variation. Kristan et al. (2019) note that the most challenging attributes, in terms of failures, are illumination change and occlusion, mirroring our findings. The latter is a challenging problem for trackers such as PBTS, although this is partly because they are not explicitly designed to handle occlusion because they are focusing on the short-term model-free tracking problem, rather than more longer-term tracking scenarios.

Although PBTS is currently unsuitable for long-term tracking problems, there are several possible adaptations that could be made to the tracker to improve its performance with respect to partial and total occlusion. TLD (Kalal, Mikolajczyk and Matas, 2012), for example, has two separate modules for tracking the object in the short-term (updated continuously) and detecting the object (updated conservatively). Having a separate detector, and only updating it when it is sure that the object is in a certain position, allows for it to be more confident as to whether an object is in the frame, and allows TLD to search for the object in subsequent frames after it has detected the object to have left the scene. Similarly, MUSTer (Hong et al., 2015b) also has short-term and long-term tracking components, that update continuously and conservatively respectively. This allows for the long-term component to redetect the object after it has been labelled as missing in previous frames. PBTS could incorporate a long-term tracking component by having an additional set of object part models that are updated more infrequently, and only when small amounts of change in the object's appearance occur between frames. This would allow the long-term component to be more robust to drift and, when combined with an increased search range from where the object was last detected, could support the search for the object in subsequent frames.

7. Summary and Conclusion

This thesis presented a feature sampling, part-based object tracking method for short-term, model-free tracking. The parts model (Section 3.3) was inspired by the approach of the pixel-wise background subtraction algorithm ViBe (Barnich and Van Droogenbroeck, 2011) and applied it to modelling object parts using sets of features to create a novel object part representation. A method of updating this new representation was also presented (Section 4.3), along with an object localisation scheme to model the globally affine, but locally non-affine, nature of tracked and deforming objects (Section 3.4). We investigated the initialisation problem (Chapter 5), proposed several solutions to it, and used the best of these as a component in a novel object part placement scheme (Section 4.1). Lastly, an ablation study was carried out (Section 6.1) on the tracker's components and tracking performance analysed across multiple datasets. The tracker was found to outperform all other part-based tracking algorithms on the VOT2018 dataset.

The part-based object model (Section 3.3) consisted of patches that were described by a sample-based representation that characterised an image patch with a set of feature samples and their corresponding counts of features that matched them in a patch. Within the tracker, our object model performed better than the standard, colour histogram, template-based approach and was found to be more robust. A novel update scheme was also introduced that updated both the features of the model (i.e. bin centres) and their respective counts. This was in contrast to template-based approaches that only update the counts of their bins. Updating the features of our model has the effect of moving them towards the features that match them in the new location, i.e. following the matching features around in colour space. This gave the model an increased level of robustness to gradual changes in the colour

of tracked objects, as shown in Section 4.3.2, and lead to an improvement in tracking performance.

Candidate locations for an object part were assessed by comparing their sets of counts with the model's set of counts using a modified version of the Bhattacharyya distance (MBD, Equation 3.2). We showed that giving a higher weighting to better matches increased performance quite considerably compared to the normal weighting of the Bhattacharyya distance. This can be easily used in other part-based methods that either take the average or product of the Bhattacharyya distance over multiple patches, e.g. ANT (Čehovin, Leonardis and Kristan, 2016a) or BMHC (Kwon and Lee, 2009). The more general idea of increasing the distance between lower quality matches, i.e. upweighting them to bias the model towards focusing on improving the lowest matching parts of their model, can be applied to a broader range of trackers. Methods such as GGTv2 (Du et al., 2017), WPCL (Zhu et al., 2015), and DPT (Lukežič, Zajc and Kristan, 2018) collectively evaluate a set of part models using a quality measure other than the Bhattacharyya distance. Therefore, we suspect that employing a similar weighting scheme to the one used here, i.e. increasing the importance of the poorer matching patches, should lead to an improvement in tracking performance.

A two-part object localisation scheme was proposed and used for locating the patches' positions in subsequent frames. This was based on observations that an object's motion between consecutive frames can generally described as being affine in nature, but with the addition of non-affine deformations within localised regions of the object. Consequentially, we generated sets of non-shearing affine transformations of the patches' previous locations and locally optimised them within a small window. These transformations were generated by sampling them from a probability distribution. However, the choice of what distribution to sample these from was non-trivial and, therefore, we investigated the amount of movement an object makes, relative to its size, in consecutive frames. It was found that object's frame-to-frame motion is distributed in a very non-Gaussian manner and is much better described by a Laplace distribution. This is because objects tend to move

small amounts relative to their size, but make much larger jumps more frequently than could be modelled by a Gaussian with the same location and length-scale of the data.

The performance of sampling candidate part locations from Gaussian and Laplace distributions was compared. A much larger length scale was needed for the Gaussian distribution than the Laplace distribution in order to achieve the same level of performance. This indicated that the Gaussian distribution would have been generating larger translations too frequently than the actual observed frequency of the translations. These results have practical implications for other trackers that generate candidate object locations to evaluate, such as BHMC (Kwon and Lee, 2009) and LGT (Čehovin, Kristan and Leonardis, 2013), as well as those that search within a fixed window, such as correlation filter-based approaches (e.g. KCF, Henriques et al., 2015). These trackers may be unintentionally causing tracking failures by making the wrong assumptions about the amount and distribution of motion an object can make between consecutive frames.

The initialisation problem also investigated in Chapter 5. It is the problem of determining which pixels, within a given bounding box, belong to the object to be tracked and which belong to its background. We described three novel methods for doing this: a sample-based background model, an alpha matting-based method, and one using a One Class SVM. Using the alpha matting technique we were able to segment the object within a given bounding box better than assuming all pixels in the object belonged to the object (as is the assumption of other techniques). As the ablation study demonstrated, using this initial segmentation as a guide to where object parts should be placed results in a considerable improvement in performance. Complementary to this, the recent work of Qin and Fan (2019) shows that all three methods improve the performance of their Siamese Network-based tracking algorithm. This demonstrates that focusing on making more use of the given bounding box is beneficial to both part-based and holistic trackers. We hope that these initial results will encourage others in the tracking community to focus more on tackling the initialisation problem as a means of improving tracking performance.

Using the aforementioned alpha-matting segmentation method, we developed a novel method to place object parts on an object, within a given bounding box, and showed that placing patches in a principled manner leads to a much higher-performing tracker. Previous part-based trackers have generally not focused on trying to differentiate between the object and its background within a bounding box. Therefore, this new type of patch placement methodology is particularly useful to other part-based techniques as it is model-agnostic and suitable for off the shelf usage. Although we have not demonstrated the performance benefits of the technique for trackers other than our own, we anticipate that this method would improve them and leave this open as an avenue for potential future work.

We also performed an investigation into patch drift and explored several methods for identifying drifting patches based on their match quality as well as their position relative to the centre of the set of patches. Unfortunately, none of the methods we investigated could consistently identify when patches were drifting. They were capable of detecting particular cases of drift, but not sufficiently well to give an increase in average tracking performance. We suspect this was due to the difficult nature of the problem and that a patch's position, in one frame, relative to all other patches is not informative enough to make a decision about whether it is drifting. In future work we would like to characterise a patch's trajectory over several frames in order to compare this to both its neighbours' trajectories and the mean trajectory of the set of object parts. This would likely allow for a more discerning drift detection scheme to be constructed as it contains a much richer representation of the recent behaviour of the patch.

Lastly, we compared our method with other part-based trackers on the VOT2018 benchmark and showed that is has substantially better performance (in terms of EAO). We highlight that our tracker is a purely part-based method and does not rely on guidance from an additional holistic tracker to guide the search process, unlike several of the other well-performing trackers, e.g. DPT (Lukežič, Zajc and Kristan, 2018) and ANT (Čehovin, Leonardis and Kristan, 2016a). In future work it would be interesting to see if using a holistic tracker to guide the search process, and limit the

region in which patches can drift to, would increase the performance of the tracker.

7.1 Future Work

We explored using different numbers and sizes of patches to represent an object. It was found that using larger numbers of smaller patches was preferable, although we suspect this is because it allows for more patches to be fitted on to smaller objects. A preliminary investigation to determine the relationship between an object's size and the optimal number and size of patches to use was carried out, but little correlation was found. Therefore, a useful extension of this work would be to investigate using patches of different sizes to represent an object. We have observed that objects tend to be more homogeneous in both colour and texture towards their centre of mass. So placing fewer, larger patches around an object's centre of mass and more, smaller patches around its outer regions might be a more useful representation than having all object parts be the same size.

In our object part model, a pixel's feature vector matches a feature sample in the model if they are within a certain distance of one another, measured by the Euclidean distance. One limitation of this scheme is that if higher dimensional features were used, e.g. 512-dimensional CNN features such as those used in HCF (Ma et al., 2015), then, due to the curse of dimensionality (Trunk, 1979), distances lose some of their discriminative ability because features are naturally further apart as the dimensionality increases. Therefore, this implies that it would be difficult to select a suitable match threshold as distances become arbitrarily large. In order to use higher dimensional features a different measure of distance or similarity to threshold would be required. One potential candidate for this is the cosine similarity as it is bounded in $[0, 1]$ for positive valued features. A future investigation into the use of more descriptive features and different similarity measures should prove a useful to further increase the tracker's performance.

The initialisation problem is far from a solved issue. We make the assumption that the object is compact, i.e. that the centre of the bounding box always contains the object, but we know, in reality, that this is not always the case, e.g. Section 5.3.3.

Recently, full scene object segmentation has become possible, in real-time, through deep neural network-based techniques such as the now famous Mask R-CNN (He et al., 2017). These are able to provide pixel-wise object labels for all objects within a scene. Using a technique similar to Mask R-CNN, but also additional prior information in the form of the given bounding box, should further improve the initial segmentation capability and therefore tracking capability in the future. We also note that part-based methods tend to have rather *ad hoc* methods of replacing patches that they identify as drifting (see Section 2.1.3) for more details. If we were able to apply the scene segmentation when a drifting object part has been identified, it would likely lead to a better placement of the part onto the object.

Tracking methods, such as the one presented in this thesis, contain many parameters (e.g. Table 6.1) that need to be set in order to produce good performance. Typically, these are tuned independently of one another or in pairs in order to assess their impact on performance. However, this makes the assumption that the tracker's parameters are independent of one another. In reality, this is extremely unlikely to be the case. For example, in our presented tracking method, the number of sets of candidate patches that need to be locally optimised may depend on the size and number of patches in the model. While this is standard practise, due to general time and computational limitations, we suggest that jointly optimising all of a tracker's parameters is a worthwhile endeavour as it may lead to uncovering previously undiscovered interactions between tracking components.

Given the number of videos that have to be processed to evaluate a dataset, e.g. 15 evaluation of each of the 60 videos in the VOT2018 benchmark, this type of computationally expensive problem naturally leads itself towards surrogate modelling. In this problem formulation a relatively cheap to evaluate, probabilistic model (e.g. a Gaussian Process) is created of an expensive function. The surrogate model is optimised to best describe the given data, e.g. taking the tracker's parameters as inputs and its corresponding EAO as outputs. This model can then be queried to find the set of parameters that will most likely, given all previous information, yield the highest performance. This set of parameters would be evaluated by the tracker

on the dataset, and the process can be repeated, with the surrogate model being retrained with the new set of parameters evaluated and its corresponding performance as a new data point. This, Bayesian, approach to optimisation would likely lead to improved tracking performance without having to exhaustively enumerate all possible parameter combinations. We direct an interested reader to the work of Snoek, Larochelle and Adams (2012), who present a practical example of using surrogate modelling to efficiently train structured Support Vector Machines and Convolutional Neural Networks, and to the review of Shahriari et al. (2016) for an extensive overview of surrogate modelling.

Bibliography

R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk (2012). 'SLIC Superpixels Compared to State-of-the-Art Superpixel Methods'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 34.11, pp. 2274–2282 (Cited on pages 80, 113 and 157).

A. Adam, E. Rivlin and I. Shimshoni (2006). 'Robust Fragments-based Tracking using the Integral Histogram'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1, pp. 798–805 (Cited on pages 14, 24, 87, 92 and 139).

O. Akin, E. Erdem, A. Erdem and K. Mikolajczyk (2016). 'Deformable Part-Based Tracking by Coupled Global and Local Correlation Filters'. In: *Journal of Visual Communication and Image Representation* 38, pp. 763–774 (Cited on pages 10, 15, 21, 24 and 99).

N. M. Artner, A. Ion and W. G. Kropatsch (2011). 'Multi-Scale 2D Tracking of Articulated Objects Using Hierarchical Spring Systems'. In: *Pattern Recognition* 44.4, pp. 800–810 (Cited on pages 10, 14 and 19).

B. Babenko, M. Yang and S. Belongie (2009). 'Visual Tracking with Online Multiple Instance Learning'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 983–990 (Cited on page 19).

B. Babenko, M. Yang and S. Belongie (2011). 'Robust Object Tracking with Online Multiple Instance Learning'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.8, pp. 1619–1632 (Cited on page 29).

S. Bai, Z. He, T. Xu, Z. Zhu, Y. Dong and H. Bai (2018). 'Multi-Hierarchical Independent Correlation Filters for Visual Tracking'. In: *CoRR* abs/1811.10302. URL: `http://arxiv.org/abs/1811.10302` (Cited on page 139).

P. Barcellos, C. Bouvié, F. L. Escouto and J. Scharcanski (2015). 'A novel Video Based System for Detecting and Counting Vehicles at User-Defined Virtual Loops'. In: *Expert Systems with Applications* 42.4, pp. 1845–1856 (Cited on page 9).

O. Barnich and M. Van Droogenbroeck (2011). 'ViBe: A Universal Background Subtraction Algorithm for Video Sequences'. In: *IEEE Transactions on Image Processing (TIP)* 20.6, pp. 1709–1724 (Cited on pages 3, 38, 41, 115 and 163).

F. Battistone, A. Petrosino and V. Santopietro (2017). 'Watch Out: Embedded Video Tracking with BST for Unmanned Aerial Vehicles'. In: *Journal of Signal Processing Systems* (Cited on pages 10, 44 and 139).

H. Bay, A. Ess, T. Tuytelaars and L. V. Gool (2008). 'Speeded-Up Robust Features (SURF)'. In: *Computer Vision and Image Understanding (CVIU)* 110.3, pp. 346–359 (Cited on page 11).

L. Bertinetto, J. Valmadre, S. Golodetz, O. Miksik and P. H. S. Torr (2016a). 'Staple: Complementary Learners for Real-Time Tracking'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 1401–1409 (Cited on pages 16 and 44).

L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi and P. H. S. Torr (2016b). 'Fully-Convolutional Siamese Networks for Object Tracking'. In: *IEEE European Conference on Computer Vision Workshop (ECCVW)*, pp. 850–865 (Cited on pages 13 and 20).

A. Bhattacharyya (1946). 'On a Measure of Divergence between Two Multinomial Populations'. In: *Sankhyā: The Indian Journal of Statistics (1933-1960)* 7.4, pp. 401–406 (Cited on pages 14, 16 and 39).

D. S. Bolme, J. R. Beveridge, B. A. Draper and Y. M. Lui (2010). 'Visual Object Tracking Using Adaptive Correlation Filters'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2544–2550 (Cited on pages 15 and 20).

A. Bosch, A. Zisserman and X. Muñoz (2006). 'Scene Classification Via pLSA'. In: *European Conference on Computer Vision (ECCV)*, pp. 517–530 (Cited on page 17).

A. Bosch, A. Zisserman and X. Muñoz (2007). 'Image Classification Using Random Forests and Ferns'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 1–8 (Cited on page 18).

Z. Cai, L. Wen, Z. Lei, N. Vasconcelos and S. Z. Li (2014). 'Robust Deformable and Occluded Object Tracking With Dynamic Graph'. In: *IEEE Transactions on Image Processing (TIP)* 23.12, pp. 5497–5509 (Cited on pages 10, 19, 20, 22, 24 and 160).

L. Čehovin, M. Kristan and A. Leonardis (2013). 'Robust Visual Tracking Using an Adaptive Coupled-Layer Visual Model'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.4, pp. 941–953 (Cited on pages 10, 14, 18, 23, 24, 44, 57, 58, 99, 105, 139 and 165).

L. Čehovin, M. Kristan and A. Leonardis (2014). 'Is My New Tracker Really Better Than Yours?' In: *IEEE Applications of Computer Vision (WACV)*, pp. 540–547 (Cited on page 29).

L. Čehovin, A. Leonardis and M. Kristan (2016a). 'Robust Visual Tracking Using Template Anchors'. In: *IEEE Applications of Computer Vision (WACV)*, pp. 1–8 (Cited on pages 10, 24, 44, 87, 139, 160, 164 and 166).

L. Čehovin, A. Leonardis and M. Kristan (2016b). 'Visual Object Tracking Performance Measures Revisited'. In: *IEEE Transactions on Image Processing (TIP)* 25.3, pp. 1261–1274 (Cited on page 25).

C. Chang and C. Lin (2011). 'LIBSVM: A Library for Support Vector Machines'. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2 (3), pp. 1–27 (Cited on page 113).

K. Chatfield, K. Simonyan, A. Vedaldi and A. Zisserman (2014). 'Return of the Devil in the Details: Delving Deep into Convolutional Nets'. In: *British Machine Vision Conference (BMVC)* (Cited on page 16).

Z. Chen, Z. Hong and D. Tao (2015). 'An Experimental Survey on Correlation Filter-based Tracking'. In: *CoRR* abs/1509.05520. URL: `https://arxiv.org/abs/1509.05520` (Cited on pages 8 and 15).

Y. Chuang, B. Curless, D. Salesin and R. Szeliski (2001). 'A Bayesian Approach to Digital Matting'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*. Vol. 2, pp. 264–271 (Cited on page 117).

R. T. Collins and M. Leordeanu (2005). 'Online Selection of Discriminative Tracking Features'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 27.10 (Cited on page 16).

D. Comaniciu, V. Ramesh and P. Meer (2000). 'Real-Time Tracking of Non-Rigid Objects Using Mean Shift'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CPVR)*. Vol. 2, pp. 142–149 (Cited on pages 10, 14, 16, 23, 39 and 40).

P. M. Corcoran, F. Nanu, S. Petrescu and P. Bigioi (2012). 'Real-Time Eye Gaze Tracking for Gaming Design and Consumer Electronics systems'. In: *IEEE Transactions on Consumer Electronics (TCE)* 58.2, pp. 347–355 (Cited on page 9).

C. Croux and P. J. Rousseeuw (1992). 'Time-Efficient Algorithms for Two Highly Robust Estimators of Scale'. In: *Computational Statistics*. Ed. by Y. Dodge and J. Whittaker. Heidelberg: Physica, pp. 411–428 (Cited on page 95).

Z. Cui, S. Xiao, J. Feng and S. Yan (2016). 'Recurrently Target-Attending Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1449–1458 (Cited on pages 13 and 20).

N. Dalal and B. Triggs (2005). 'Histograms of Oriented Gradients for Human Detection'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*. Vol. 1, pp. 886–893 (Cited on pages 9, 16, 18, 76 and 142).

M. Danelljan, G. Bhat, F. S. Khan and M. Felsberg (2017). 'ECO: Efficient Convolution Operators for Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6931–6939 (Cited on pages 20 and 139).

M. Danelljan, G. Häger, F. S. Khan and M. Felsberg (2015a). 'Convolutional Features for Correlation Filter Based Visual Tracking'. In: *IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 621–629 (Cited on pages 16 and 156).

M. Danelljan, G. Häger, F. S. Khan and M. Felsberg (2015b). 'Learning Spatially Regularized Correlation Filters for Visual Tracking'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 4310–4318 (Cited on pages 10, 20, 23 and 58).

M. Danelljan, F. S. Khan, M. Felsberg and J. van de Weijer (2014a). 'Adaptive Color Attributes for Real-Time Visual Tracking'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 1090–1097 (Cited on pages 10, 15, 16, 41 and 58).

M. Danelljan, A. Robinson, F. S. Khan and M. Felsberg (2016). 'Beyond Correlation Filters: Learning Continuous Convolution Operators for Visual Tracking'. In: *IEEE European Conference on Computer Vision (ECCV)*, pp. 472–488 (Cited on pages 10 and 58).

M. Danelljan, G. Häger, F. Shahbaz Khan and M. Felsberg (2014b). 'Accurate Scale Estimation for Robust Visual Tracking'. In: *British Machine Vision Conference (BMVC)* (Cited on page 16).

G. De Ath and R. Everson (2018). 'Visual Object Tracking: The Initialisation Problem'. In: *Conference on Computer and Robot Vision (CRV)*, pp. 142–149 (Cited on pages 2, 6 and 130).

T. B. Dinh, N. Vo and G. Medioni (2011). 'Context Tracker: Exploring Supporters and Distracters in Unconstrained Environments'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 1177–1184 (Cited on page 148).

D. Du, H. Qi, W. Li, L. Wen, Q. Huang and S. Lyu (2016). 'Online Deformable Object Tracking Based on Structure-Aware Hyper-Graph'. In: *IEEE Transactions on Image Processing (TIP)* 25.8, pp. 3572–3584 (Cited on pages 10, 22 and 24).

D. Du, H. Qi, L. Wen, Q. Tian, Q. Huang and S. Lyu (2017). 'Geometric Hypergraph Learning for Visual Tracking'. In: *IEEE Transactions on Cybernetics (TCYB)* PP.99, pp. 1–14 (Cited on pages 10, 22, 24, 103, 114, 160 and 164).

M. Fairchild (2005). *Color Appearance Models*. The Wiley-IS&T Series in Imaging Science and Technology. Wiley (Cited on page 16).

H. Fan and H. Ling (2017). 'SANet: Structure-Aware Network for Visual Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, pp. 2217–2224 (Cited on page 13).

H. Fan and J. Xiang (2017). 'Robust Visual Tracking via Local-Global Correlation Filter'. In: *AAAI Conference on Artificial Intelligence*, pp. 4025–4031 (Cited on pages 21 and 24).

M. A. Fischler and R. C. Bolles (1981). 'Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography'. In: *Communications of the ACM* 24.6, pp. 381–395 (Cited on page 12).

K. Fragkiadaki, S. Levine, P. Felsen and J. Malik (2015). 'Recurrent Network Models for Human Dynamics'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 4346–4354 (Cited on page 9).

H. K. Galoogahi, A. Fagg, C. Huang, D. Ramanan and S. Lucey (2017). 'Need for Speed: A Benchmark for Higher Frame Rate Object Tracking'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 1134–1143 (Cited on page 25).

H. K. Galoogahi, T. Sim and S. Lucey (2013). 'Multi-channel Correlation Filters'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 3072–3079 (Cited on page 15).

R. Girshick, J. Donahue, T. Darrell and J. Malik (2014). 'Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 580–587 (Cited on page 13).

H. Grabner, M. Grabner and H. Bischof (2006). 'Real-Time Tracking via On-Line Boosting'. In: *British Machine Vision Conference (BMVC)* (Cited on page 149).

F. R. Hampel (1974). 'The Influence Curve and Its Role in Robust Estimation'. In: *Journal of the American Statistical Association* 69.346, pp. 383–393 (Cited on page 94).

D. W. Hansen and Q. Ji (2010). 'In the Eye of the Beholder: A Survey of Models for Eyes and Gaze'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 32.3, pp. 478–500 (Cited on page 9).

S. Hare, S. Golodetz, A. Saffari, V. Vineet, M. M. Cheng, S. L. Hicks and P. H. S. Torr (2016). 'Struck: Structured Output Tracking with Kernels'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 38.10, pp. 2096–2109 (Cited on pages 10, 19, 44 and 148).

K. He, G. Gkioxari, P. Dollár and R. Girshick (2017). 'Mask R-CNN'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988 (Cited on page 168).

S. He, Q. Yang, R. W. H. Lau, J. Wang and M. Yang (2013). 'Visual Tracking via Locality Sensitive Histograms'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2427–2434 (Cited on page 14).

E. Hellinger (1909). 'Neue Begründung der Theorie Quadratischer Formen von Unendlichvielen Veränderlichen.' In: *Journal für die Reine und Angewandte Mathematik* 136, pp. 210–271 (Cited on page 40).

J. F. Henriques, R. Caseiro, P. Martins and J. Batista (2015). 'High-Speed Tracking with Kernelized Correlation Filters'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 37.3, pp. 583–596 (Cited on pages 2, 10, 11, 15, 16, 20, 27, 58, 139, 161 and 165).

J. F. Henriques, R. Caseiro, P. Martins and J. Batista (2012). 'Exploiting the Circulant Structure of Tracking-by-Detection with Kernels'. In: *IEEE European Conference on Computer Vision (ECCV)*, pp. 702–715 (Cited on page 149).

S. Hong, T. You, S. Kwak and B. Han (2015a). 'Online Tracking by Learning Discriminative Saliency Map with Convolutional Neural Network'. In: *International Conference on Machine Learning (ICML)*, pp. 597–606 (Cited on page 13).

Z. Hong, C. Wang, X. Mei, D. Prokhorov and D. Tao (2015b). 'MUlti-Store Tracker (MUSTer): A Cognitive Psychology Inspired Approach to Object Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 749–758 (Cited on pages 12 and 162).

D. Huang, C. Shan, M. Ardabilian, Y. Wang and L. Chen (2011). 'Local Binary Patterns and Its Application to Facial Image Analysis: A Survey'. In: *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)* 41.6, pp. 765–781 (Cited on page 17).

A. Jazayeri, H. Cai, J. Y. Zheng and M. Tuceryan (2011). 'Vehicle Detection and Tracking in Car Video Based on Motion Model'. In: *IEEE Transactions on Intelligent Transportation Systems (TITS)* 12.2, pp. 583–595 (Cited on page 9).

X. Jia, H. Lu and M. Yang (2012). 'Visual Tracking via Adaptive Structural Local Sparse Appearance Model'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 1822–1829 (Cited on page 148).

J. Johnander, M. Danelljan, F. S. Khan and M. Felsberg (2017). 'DCCO: Towards Deformable Continuous Convolution Operators for Visual Tracking'. In: *International Conference on Computer Analysis of Images and Patterns*, pp. 55–67 (Cited on pages 24 and 105).

Z. Kalal, K. Mikolajczyk and J. Matas (2012). 'Tracking-Learning-Detection'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 34.7, pp. 1409–1422 (Cited on pages 148 and 162).

A. R. Kosiorek, A. Bewley and I. Posner (2017). 'Hierarchical Attentive Recurrent Tracking'. In: *Conference on Neural Information Processing Systems (NIPS)* (Cited on page 13).

M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, Z. Luka Čehovin, T. Vojír, G. Häger, A. Lukežič, G. Fernandez Dominguez et al. (2016a). 'The Visual Object Tracking VOT2016 Challenge Results'. In: *IEEE European Conference on Computer Vision (ECCV)*, pp. 777–823 (Cited on pages 5, 10, 28, 34, 35, 111, 121, 128, 138 and 147).

M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. Č. Zajc, T. Vojír, G. Bhat, A. Lukežič, A. Eldesokey et al. (2019). 'The Visual Object Tracking VOT2018 Challenge Results'. In: *IEEE European Conference on Computer Vision Workshop (ECCVW)*, pp. 3–53 (Cited on pages 4, 5, 8, 10, 12, 15, 16, 25, 27, 28, 132, 134, 139, 140, 160 and 162).

M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. Č. Zajc, T. Vojír, G. Häger, A. Lukežič, A. Eldesokey et al. (2017). 'The Visual Object Tracking VOT2017 Challenge Results'. In: *IEEE International Conference on Computer Vision Workshop (ICCVW)* (Cited on page 27).

M. Kristan, J. Matas, A. Leonardis, M. Felsberg, Z. Luka Čehovin, G. Fernández, T. Vojír, G. Häger, G. Nebehay, R. Pflugfelder et al. (2015). 'The Visual Object Tracking VOT2015 Challenge Results'. In: *IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 564–586 (Cited on pages 1, 9, 12, 27, 28, 35 and 110).

M. Kristan, J. Matas, A. Leonardis, T. Vojír, R. Pflugfelder, G. Fernández, G. Nebehay, F. Porikli and L. Č. Zajc (2016b). 'A Novel Performance Evaluation Methodology for Single-Target Trackers'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 38.11, pp. 2137–2155 (Cited on pages 8 and 27).

M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, Z. Luka Čehovin, G. Nebehay, G. Fernandez Dominguez and T. Vojír (2013). 'The Visual Object Tracking VOT2013 Challenge Results'. In: *IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 98–111 (Cited on pages 1, 3, 8, 12 and 25).

A. Krizhevsky, I. Sutskever and G. E. Hinton (2012). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Conference on Neural Information Processing Systems (NIPS)*, pp. 1097–1105 (Cited on page 13).

J. Kwon and K. M. Lee (2009). 'Tracking of a Non-Rigid Object via Patch-Based Dynamic Appearance Modeling and Adaptive Basin Hopping Monte Carlo Sampling'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 1208–1215 (Cited on pages 10, 14, 21, 24, 87, 99, 160, 164 and 165).

J. Kwon and K. M. Lee (2010). 'Visual Tracking Decomposition'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 1269–1276 (Cited on page 149).

J. Kwon and K. M. Lee (2011). 'Tracking by Sampling Trackers'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 1195–1202 (Cited on page 149).

A. Li, M. Lin, Y. Wu, M. Yang and S. Yan (2016). 'NUS-PRO: A New Visual Tracking Challenge'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 38.2, pp. 335–349 (Cited on page 25).

B. Li, J. Yan, W. Wu, Z. Zhu and X. Hu (2018). 'High Performance Visual Tracking With Siamese Region Proposal Network'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)* (Cited on pages 139 and 158).

X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick and A. V. D. Hengel (2013). 'A Survey of Appearance Models in Visual Object Tracking'. In: *IEEE Transactions on Intelligent Transportation Systems (TITS)* 4.4:58, pp. 1–48 (Cited on pages 8, 14 and 16).

P. Liang, E. Blasch and H. Ling (2015). 'Encoding Color Information for Visual Tracking: Algorithms and Benchmark'. In: *IEEE Transactions on Image Processing (TIP)* 24.12, pp. 5630–5644 (Cited on page 25).

B. Liu, J. Huang, C. Kulikowski and L. Yang (2013). 'Robust Visual Tracking Using Local Sparse Appearance Model and K-Selection'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.12, pp. 2968–2981 (Cited on page 149).

F. T. Liu, K. M. Ting and Z. H. Zhou (2008). 'Isolation Forest'. In: *IEEE International Conference on Data Mining (ICDM)*, pp. 413–422 (Cited on page 125).

S. Liu, T. Zhang, X. Cao and C. Xu (2016). 'Structural Correlation Filter for Robust Visual Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4312–4320 (Cited on pages 15, 21 and 24).

T. Liu, G. Wang and Q. Yang (2015). 'Real-Time Part-Based Visual Tracking via Adaptive Correlation Filters'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4902–4912 (Cited on pages 11, 15, 21, 24 and 99).

T. Liu, G. Wang, Q. Yang and L. Wang (2018). 'Part-Based Tracking via Discriminative Correlation Filters'. In: *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)* (Cited on page 15).

D. G. Lowe (1999). 'Object Recognition from Local Scale-Invariant Features'. In: *IEEE International Conference on Computer Vision (ICCV)*. Vol. 2, pp. 1150–1157 (Cited on pages 11, 12, 17, 76 and 114).

B. D. Lucas and T. Kanade (1981). 'An Iterative Image Registration Technique with an Application to Stereo Vision'. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 2, pp. 674–679 (Cited on page 24).

A. Lukežič, L. Č. Zajc and M. Kristan (2018). 'Deformable Parts Correlation Filters for Robust Visual Tracking'. In: *IEEE Transactions on Cybernetics (TCYB)*, pp. 1–13 (Cited on pages 10, 14, 15, 22, 24, 58, 139, 164 and 166).

A. Lukežič, T. Vojíř, L. Čehovin Zajc, J. Matas and M. Kristan (2018). 'Discriminative Correlation Filter Tracker with Channel and Spatial Reliability'. In: *International Journal of Computer Vision* 126.7, pp. 671–688 (Cited on pages 16 and 20).

C. Ma, J. Huang, X. Yang and M. Yang (2015). 'Hierarchical Convolutional Features for Visual Tracking'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 3074–3082 (Cited on pages 20 and 167).

Y. Ma, X. Chen and G. Chen (2011). 'Pedestrian Detection and Tracking Using HOG and Oriented-LBP Features'. In: *Network and Parallel Computing*, pp. 176–184 (Cited on page 16).

P. Majaranta and A. Bulling (2014). 'Eye Tracking and Eye-Based Human-Computer Interaction'. In: *Advances in Physiological Computing*. Ed. by S. H. Fairclough and K. Gilleade. London: Springer, pp. 39–65 (Cited on page 9).

M. E. Maresca and A. Petrosino (2013). 'MATRIOSKA: A Multi-Level Approach to Fast Tracking by Learning'. In: *Image Analysis and Processing (ICIAP)*. Springer Berlin Heidelberg, pp. 419–428 (Cited on pages 10, 12, 19 and 139).

M. E. Maresca and A. Petrosino (2015). 'Clustering Local Motion Estimates for Robust and Efficient Object Tracking'. In: *IEEE European Conference on Computer Vision (ECCV)*, pp. 244–253 (Cited on pages 10 and 139).

L. Matthews, T. Ishikawa and S. Baker (2004). 'The Template Update Problem'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 26.6, pp. 810–815 (Cited on page 18).

M. D. McKay, R. J. Beckman and W. J. Conover (1979). 'A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code'. In: *Technometrics* 21.2, pp. 239–245 (Cited on pages 66 and 68).

X. Mei and H. Ling (2011). 'Robust Visual Tracking and Vehicle Classification via Sparse Representation'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 33.11, pp. 2259–2272 (Cited on page 9).

T. B. Moeslund, A. Hilton and V. Krüger (2006). 'A Survey of Advances in Vision-Based Human Motion Capture and Analysis'. In: *Computer Vision and Image Understanding (CVIU)* 104.2, pp. 90–126 (Cited on page 8).

A. Montero, J. Lang and R. Laganière (2015). 'Scalable Kernel Correlation Filter with Sparse Feature Integration'. In: *IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 587–594 (Cited on page 12).

H. Nam and B. Han (2016). 'Learning Multi-Domain Convolutional Neural Networks for Visual Tracking'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 4293–4302 (Cited on pages 10, 13, 20 and 158).

W. Nam, P. Dollár and J. H. Han (2014). 'Local Decorrelation for Improved Pedestrian Detection'. In: *Conference on Neural Information Processing Systems (NIPS)*. Vol. 1, pp. 424–432 (Cited on page 9).

G. Nebehay and R. Pflugfelder (2015). 'Clustering of Static-Adaptive Correspondences for Deformable Object Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2784–2791 (Cited on page 12).

K. Nummiaro, E. Koller-Meier and L. V. Gool (2003). 'An Adaptive Color-Based Particle Filter'. In: *Image and Vision Computing* 21.1, pp. 99–110 (Cited on pages 16 and 19).

T. Ojala, M. Pietikäinen and T. Mäenpää (2002). 'Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns'. In:

IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 24.7, pp. 971–987 (Cited on pages 16, 17, 76, 114 and 123).

S. Ojha and S. Sakhare (2015). 'Image Processing Techniques for Object Tracking in Video Surveillance - A Survey'. In: *International Conference on Pervasive Computing (ICPC)*, pp. 1–6 (Cited on page 8).

W. Ouyang and X. Wang (2013). 'Joint Deep Learning for Pedestrian Detection'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 2056–2063 (Cited on page 9).

F. Porikli (2005). 'Integral Histogram: a Fast Way to Extract Histograms in Cartesian Spaces'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 829–836 (Cited on page 45).

F. Porikli (2006). 'Achieving Real-Time Object Detection and Tracking Under Extreme Conditions'. In: *Journal of Real-time Image Processing* 1.1, pp. 33–40 (Cited on page 8).

X. Qin and Z. Fan (2019). 'Initial Matting-Guided Visual Tracking With Siamese Network'. In: *IEEE Access* 7, pp. 41669–41677 (Cited on pages 130 and 165).

A. S. Razavian, H. Azizpour, J. Sullivan and S. Carlsson (2014). 'CNN Features Off-the-Shelf: An Astounding Baseline for Recognition'. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, pp. 512–519 (Cited on page 12).

J. Redmon, S. Divvala, R. Girshick and A. Farhadi (2016). 'You Only Look Once: Unified, Real-Time Object Detection'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788 (Cited on page 12).

J. Redmon and A. Farhadi (2017). 'YOLO9000: Better, Faster, Stronger'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525 (Cited on page 12).

S. Ren, K. He, R. Girshick and J. Sun (2015). 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks'. In: *Conference on Neural Information Processing Systems (NIPS)*, pp. 91–99 (Cited on page 12).

X. Ren and J. Malik (2003). 'Learning a Classification Model for Segmentation'. In: *IEEE International Conference on Computer Vision (ICCV)*. Vol. 1, pp. 10–17 (Cited on pages 22 and 79).

P. J. Rousseeuw and C. Croux (1993). 'Alternatives to the Median Absolute Deviation'. In: *Journal of the American Statistical Association* 88.424, pp. 1273–1283 (Cited on pages 94 and 96).

P. J. Rousseeuw and A. M. Leroy (1987). *Robust Regression and Outlier Detection*. Wiley Series in Probability and Statistics. Wiley (Cited on page 94).

E. Rublee, V. Rabaud, K. Konolige and G. Bradski (2011). 'ORB: An Efficient Alternative to SIFT or SURF'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 2564–2571 (Cited on page 11).

Y. Rubner, C. Tomasi and L. J. Guibas (1998). 'A Metric for Distributions with Applications to Image Databases'. In: *International Conference on Computer Vision (ICCV)*, pp. 59–66 (Cited on page 107).

S. Salti, A. Cavallaro and L. D. Stefano (2012). 'Adaptive Appearance Modeling for Video Tracking: Survey and Evaluation'. In: *IEEE Transactions on Image Processing (TIP)* 21.10, pp. 4334–4348 (Cited on page 8).

B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola and R. C. Williamson (2001). 'Estimating the Support of a High-Dimensional Distribution'. In: *Neural Computation* 13.7, pp. 1443–1471 (Cited on pages 5 and 112).

B. Schölkopf and A. J. Smola (2001). *Learning with Kernels*. Cambridge, MA, USA: MIT Press (Cited on page 113).

P. Senna, I. N. Drummond and G. S. Bastos (2017). 'Real-Time Ensemble-Based Tracker with Kalman Filter'. In: *Conference on Graphics, Patterns and Images*. SIBGRAPI, pp. 338–344 (Cited on page 58).

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams and N. de Freitas (2016). 'Taking the Human Out of the Loop: A Review of Bayesian Optimization'. In: *Proceedings of the IEEE* 104.1, pp. 148–175 (Cited on page 169).

P. Simon and U. Vijayasundaram (2018). 'Review of Texture Descriptors for Texture Classification'. In: *Data Engineering and Intelligent Computing*. Singapore: Springer Singapore, pp. 159–176 (Cited on page 16).

K. Simonyan and A. Zisserman (2015). 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: *International Conference on Learning Representations (ICLR)* (Cited on page 13).

A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan and M. Shah (2014). 'Visual Tracking: An Experimental Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 36.7, pp. 1442–1468 (Cited on pages 8 and 25).

A. R. Smith (1978). 'Color Gamut Transform Pairs'. In: *ACM Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH. New York, NY, USA: ACM, pp. 12–19 (Cited on pages 17 and 42).

J. Snoek, H. Larochelle and R. P. Adams (2012). 'Practical Bayesian Optimization of Machine Learning Algorithms'. In: *Conference on Neural Information Processing Systems (NIPS)*, pp. 2951–2959 (Cited on page 169).

Y. Song, C. Ma, L. Gong, J. Zhang, R. Lau and M.-H. Yang (2017). 'CREST: Convolutional Residual Learning for Visual Tracking'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 2555–2564 (Cited on pages 13 and 20).

D. Stutz, A. Hermans and B. Leibe (2018). 'Superpixels: An Evaluation of the State-of-the-Art'. In: *Computer Vision and Image Understanding (CVIU)* 166, pp. 1–27 (Cited on pages 79 and 81).

Y. Sui, Y. Tang and L. Zhang (2015). 'Discriminative Low-Rank Tracking'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 3002–3010 (Cited on page 19).

C. Sun, D. Wang, H. Lu and M.-H. Yang (2018). 'Learning Spatial-Aware Regressions for Visual Tracking'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)* (Cited on pages 10, 13 and 58).

V. Takala and M. Pietikainen (2007). 'Multi-Object Tracking Using Color, Texture and Motion'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Cited on page 16).

D. M. Tax and R. P. Duin (2004). 'Support Vector Data Description'. In: *Machine Learning* 54.1, pp. 45–66 (Cited on page 113).

G. V. Trunk (1979). 'A Problem of Dimensionality: A Simple Example'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 1.3, pp. 306–307 (Cited on page 167).

J. van de Weijer, C. Schmid, J. Verbeek and D. Larlus (2009). 'Learning Color Names for Real-World Applications'. In: *IEEE Transactions on Image Processing (TIP)* 18.7, pp. 1512–1523 (Cited on pages 16 and 130).

A. Vedaldi, V. Gulshan, M. Varma and A. Zisserman (2009). 'Multiple Kernels for Object Detection'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 606–613 (Cited on page 18).

T. Vojíř and J. Matas (2014). 'The Enhanced Flock of Trackers'. In: *Registration and Recognition in Images and Videos*. Ed. by R. Cipolla, S. Battiato and G. M. Farinella. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 113–136 (Cited on pages 27 and 139).

T. Vojíř and J. Matas (2017). *Pixel-Wise Object Segmentations for the VOT 2016 Dataset*. Research Report CTU-CMP-2017-01. Prague, Czech Republic: Center for Machine Perception, Czech Technical University (Cited on pages 2, 72, 110 and 121).

T. Vojíř, J. Noskova and J. Matas (2013). 'Robust Scale-Adaptive Mean-Shift for Tracking'. In: *Scandinavian Conference on Image Analysis (SCIA)*, pp. 652–663 (Cited on page 27).

L. Wang, W. Ouyang, X. Wang and H. Lu (2016). 'STCT: Sequentially Training Convolutional Networks for Visual Tracking'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1373–1381 (Cited on page 13).

Q. Wang, L. Zhang, L. Bertinetto, W. Hu and P. H. S. Torr (2018). 'Fast Online Object Tracking and Segmentation: A Unifying Approach'. In: *CoRR* abs/1812.05050. URL: http://arxiv.org/abs/1812.05050 (Cited on page 147).

X. Wang, G. Hua and T. X. Han (2010). 'Discriminative Tracking by Metric Learning'. In: *European Conference on Computer Vision (ECCV)*, pp. 200–214 (Cited on pages 18 and 19).

Y. Wu, J. Lim and M.-H. Yang (2013). 'Online Object Tracking: A Benchmark.' In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 2411–2418 (Cited on pages 1, 9, 10, 12, 25 and 110).

Y. Wu, J. Lim and M.-H. Yang (2015). 'Object Tracking Benchmark'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 37.9, pp. 1834–1848 (Cited on pages 5, 25, 131, 147 and 148).

J. Xiao, R. Stolkin and A. Leonardis (2015). 'Single Target Tracking Using Adaptive Clustered Decision Trees and Dynamic Multi-Level Appearance Models'. In: *IEEE Computer Vision and Pattern Recognition (CVPR)*, pp. 4978–4987 (Cited on pages 10, 19, 24, 44 and 58).

T. Xu, Z. Feng, X. Wu and J. Kittler (2018). 'Learning Adaptive Discriminative Correlation Filters via Temporal Consistency Preserving Spatial Feature Selection for Robust Visual Tracking'. In: *CoRR* abs/1807.11348. URL: http://arxiv.org/abs/1807.11348 (Cited on pages 139 and 158).

Y. Xu, J. Dong, B. Zhang and D. Xu (2016). 'Background Modelling Methods in Video Analysis: A Review and Comparative Evaluation'. In: *CAAI Transactions on Intelligence Technology* 1.1, pp. 43–60 (Cited on page 3).

H. Yang, L. Shao, F. Zheng, L. Wang and Z. Song (2011). 'Recent Advances and Trends in Visual Tracking: A Review'. In: *Neurocomputing* 74.18, pp. 3823–3831 (Cited on page 8).

R. Yao, S. Xia, Z. Zhang and Y. Zhang (2017). 'Real-Time Correlation Filter Tracking by Efficient Dense Belief Propagation With Structure Preserving'. In: *IEEE Transactions on Multimedia (TMM)* 19.4, pp. 772–784 (Cited on pages 15 and 24).

K. M. Yi, H. Jeong, S. W. Kim, S. Yin, S. Oh and J. Y. Choi (2015). 'Visual Tracking of Non-Rigid Objects with Partial Occlusion Through Elastic Structure of Local Patches and Hierarchical Diffusion'. In: *Image and Vision Computing (IVC)* 39, pp. 23–37 (Cited on pages 10, 14, 19, 22, 23, 24, 44, 57, 58, 103 and 105).

A. Yilmaz, O. Javed and M. Shah (2006). 'Object Tracking: A Survey'. In: *ACM Computing Surveys* 38.4 (Cited on pages 1 and 8).

G. Yuan, Y. Gao and D. Xu (2011). 'A Moving Objects Tracking Method Based on a Combination of Local Binary Pattern Texture and Hue'. In: *Procedia Engineering (CEIS 2011)* 15, pp. 3964–3968 (Cited on page 16).

L. Zhang, J. Varadarajan, P. N. Suganthan, N. Ahuja and P. Moulin (2017). 'Robust Visual Tracking Using Oblique Random Forests'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5825–5834 (Cited on page 19).

S. Zhang, H. Yao, X. Sun and X. Lu (2013). 'Sparse Coding Based Visual Tracking: Review and Experimental Comparison'. In: *Pattern Recognition* 46.7, pp. 1772–1788 (Cited on page 8).

Y. Zheng and C. Kambhamettu (2009). 'Learning Based Digital Matting'. In: *IEEE International Conference on Computer Vision (ICCV)*, pp. 889–896 (Cited on pages 6, 117, 118, 119, 123, 128, 129 and 130).

G. Zhu, J. Wang, C. Zhao and H. Lu (2015). 'Weighted Part Context Learning for Visual Tracking'. In: *IEEE Transactions on Image Processing (TIP)* 24.12, pp. 5140–5151 (Cited on pages 10, 22, 160 and 164).

Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan and W. Hu (2018). 'Distractor-Aware Siamese Networks for Visual Object Tracking'. In: *IEEE European Conference on Computer Vision (ECCV)*, pp. 103–119 (Cited on page 160).