Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Scale Estimation in Visual Object Tracking

Bachelorthesis
at Research Group Knowledge Technology, WTM
Prof. Dr. Stefan Wermter

Department of Informatics
MIN-Faculty
Universität Hamburg

submitted by
Finn Rietz
Course of study: Human-Computer Interaction
Matrikelnr.: 6799896
on
9.4.2019

Examiners: Dr. Stefan Heinrich
Dr. Matthias Kerzel

Abstract

Robust visual object tracking is a key requirement in the field of robotics, as it enables advanced robot environment interaction. In the past, Convolutional Neural Networks (CNNs) have been shown to be powerful feature extractors and are viable for visual object tracking, based on the extracted features. While most state-of-the-art trackers achieve strong results in controlled settings, significant scale variations of the target object during tracking poses a challenging problem, because it requires online learning of new visual features, corresponding to the object at a different scale. As a consequence, scale variation requires specific and sophisticated care.

This thesis provides thorough analysis of the available algorithms for handling scale variations, from which two algorithms have been carefully selected and implemented in the HIOB tracking framework. Both algorithms are extended to support independent scaling across the x and y-axis, additionally, update strategies are developed that control on which frames the algorithms are executed. The TB100 dataset is used for evaluation on a broad and diverse range of sequence. Additionally, evaluation on the NICO dataset reveals performance of the two algorithms when facing typical robot environment interaction challenges. The results show strong scale estimation capabilities of one algorithm, while the second algorithm that has been developed shows promising potential.

Zusammenfassung

Das Verfolgen von Objekten über mehrere Bilder hinweg ist eine grundlegende Voraussetzung in der Robotik, da es fortgeschrittene Roboter-Umwelt Interaktion ermöglicht. In der Vergangenheit wurde gezeigt, dass Convolutional Neural Networks in der Lage sind visuelle Merkmale zu extrahieren, basierend auf welchen, Objekte verfolgt werden können. Während viele Tracker in kontrollierten Umgebungen gute Ergebnisse erzielen, stellen signifikante Änderungen der Größe des verfolgten Objekts ein anspruchsvolles Problem dar, da online neue Merkmale gelernt werden müssen, welche das Objekt in veränderter Größe beschreiben. Daher wird eine Lösung für das spezifische Problem der sich verändernden Größe benötigt.

Diese Bachelorarbeit stellt eine sorgfältige Untersuchung der vorhanden Algorithmen, welche die Problematik der sich verändernden Größe thematisieren, zur Verfügung. Basierend auf dieser, wurden zwei Algorithmen ausgewählt und in dem HIOB Tracking System implementiert. Beide Algorithmen wurde erweitert, wodurch unabhängige Skalierung auf der x und y-Achse möglich ist. Außerdem wurde Aktualisierungs-Stratgien entwickelt, welche steuern, auf welchen Bildern die Algorithmen ausgeführt werden. Der TB100 Datensatz wird benutzt um die Ergebnisse der Algorithmen auf einer diversen Sammlung von Sequenzen zu untersuchen, wohingegen die Ergebnisse auf dem NICO Datensatz zeigen, wie sich die unterschiedlichen Algorithmen verhalten, wenn sie den typischen Schwierigkeiten der Roboter-Umwelt Interaktion ausgesetzt sind. Die erzielten Ergebnisse deuten gute Ergebnisse in der Abschätzung der Größe von einem der beiden Algorithmen an, wohingegen der zweite Algorithmus, welcher entwickelt wurde, vielversprechendes Potential zeigt.

Contents

Chapter 1

Introduction

Visual Object Tracking is a popular yet challenging topic in the field of Computer Vision and is a fundamental requirement for advanced Computer Vision Tasks [35]. Being able to successfully track objects based on visual data only, enables tackling of higher level goals related to computer vision. Autonomous agents or vehicles need to be able to keep track of pedestrians, other agents or objects in general [23]. Other tasks like the automatic evaluation of security footage [27] or general robotics and human-robot interaction require successful visual tracking performance. For research groups related to the fields of robotics and computer vision it is, thus, highly desirable to be able to achieve robust tracking performance, because being able to accurately track objects in a real-world environment is a key requirement for any autonomous agent.

Most up to date visual object trackers report precise and accurate results on ideal tracking sequences [28], however difficulties arise with varying conditions like In-Plane Rotation (IPR) of the object, Illumination Variations in the scene (IV) or Occlusion (OCC) of the object during tracking, which are not trivial challenges to overcome for a visual tracking algorithm. What most of these conditions have in common is that they can cause the target object to greatly change its appearance between two frames due to the occurrence of one of the described conditions. While most of those factors do not pose any problem for human vision, they pose great challenges for various image processing algorithms (including CNNs) and require careful and sophisticated handling.

Another challenging and naturally occurring condition is Scale Variation (SV) of the object. SV tends to occur when the viewpoint of the tracker is not static, for example when an agent is moving through 3D space, but it can also be caused by static agents that are located in a dynamic environment. In order to maintain a precise model representation of the object, the tracker needs to adapt to the scale change and learn the representation of the object at a new scale. The process of detecting and handling the scale change is referred to as Scale Estimation (SE), which poses the main focus of this bachelor thesis.

1.1 Research objective

As with most of the conditions mentioned earlier, SV requires a specific solution in order to ensure robust tracking results. Thus, it is required to find an algorithm specifically tailored to handle the task of estimating the scale of an object during tracking. The **HI**erarchical **OB**ject tracker *HIOB* has been developed by Peer Springstübe and originally has no solution for SE and maintains a static bounding box, based on the initial size of the object [30]. The HIOB tracker finds active use in combination with the **N**euro-**I**nspired **CO**mpanion *NICO* by Kerzel et al. and serves as a research framework for developmental robotics, which highlights the relevance of the framework [19, 17]. The HIOB tracker poses the foundation for this thesis and will be expanded upon. Thus, the first objective of this bachelor thesis is to analyze the existing algorithms for SE and find a viable approach under considerations of the requirements and constraints that arise in the developmental robotics use case. The research question is formulated as follows:

What is a fitting approach for the problem of estimating the changing scale of the target object during tracking, under consideration of the developmental robotics constraints?

To find an answer for this question, an analysis of the available algorithms is conducted to find generally valid and stat-of-the-art algorithms, which deal with the target object changing its size during tracking. The following questions will be discussed in greater detail throughout this thesis because they are closely related to the main research questions:

- What are the main challenges in scale estimation?
- What approaches are adequate for the HIOB framework in combination the NICO robot as a development robotics research platform?

The positive and negative consequences of the implementation of a module that specifically handles SV during tracking will be analyzed and evaluated.

1.2 Organization of this thesis

The remainder of this thesis is structured as follows: In chapter 2, a basic overview of the underling technologies and principles is provided. This includes the introduction to Convolutional Neural Networks (CNNs) and more specifically CNN based object tracking, as HIOB makes heavy use of the CNN technology [30]. Additionally, an introduction to correlation filtering will be given, as one of the selected and implemented algorithms is based on correlation filtering. General related work, the HIOB framework, and existing algorithms for scale estimation are introduced and discussed in chapter 3. How the problem of SE has been dealt with, what datasets are used in this thesis, and how the different approaches have been implemented

is described in chapter 4. Chapter 5 presents the final results and findings of this thesis, evaluates the different implemented approaches, and provides an in depth analysis of the behavior of the different algorithm on exemplary, representative sequences. Finally, chapter 6 provides a discussion of the results and findings, under consideration of the main research question, and provides possible future work topics.

Chapter 2

Basics

This chapter provides introductions into the theoretical concepts that will be needed to develop a solid understanding of the different approaches that will be introduced later. Specifically, the concepts and principles that are used in the algorithms that have been implemented will be explained here.

2.1 Defining the tracking task

Before we explore the theoretical concepts that are of critical importance for the algorithms discussed in chapter 3, it makes sense to formally define what is to be understood as the overall goal of tracking. This is necessary because tracking can occur in different contexts. It needs to be discriminated between tracking in an actual autonomous agent and tracking on prerecorded image sequence. This differentiation is necessary, because an agent autonomously navigating through 3D-space needs to be able to detect each object in order to track it, while this step of detection can be ignored for tracking on a prerecorded image sequence.

In general accordance with this, a first categorization of tracking algorithms is provided by Kalal et al. [18], where a distinction is made between tracking and detection approaches:

- Detection approaches: Detection approaches are applied for every frame and estimate the position of the target object without considering the previous position. While detection approaches do not drift off the target because they do not accumulate errors (e.g. in the representation model) over time, they require offline training and can thus not be applied to unknown objects.

- Tracking approaches: The definition of a tracking approach is that it only requires initialization once and from the point of initialization on, a tracking algorithm tries to estimate the objects motion. This makes tracking approaches fast, as tracking eliminates the need for repeated detection and is also less costly than running a detector on every frame but also makes them prone to accumulate errors over time, as the object can change its appearance drastically [30, 3]. Trackers also tend to fail when the object completely

leaves the viewport and reappears, possible rotated on multiple axis and thus possibly having a vastly changed appearance.

Kalal et al. offer definitions for both tracking and detection tasks [18]. Tracking is defined as "the task of estimating the objects movement" and detection is defined as "the task of localizing object's in an input image". While this is a solid foundation of a definition for the tracking task, the definitions are vaguely formulated and a more extensive definition is provided in the following section.

Kalal et al. further differentiate between *static* and *adaptive* tracking approaches. In static approaches, the template/model representation of the target object is not changed during tracking, assuming that the appearance of the object won't change during tracking. In adaptive approaches, the template/model adapts to changes in appearance of the object. Adapting to the changes of appearance of the target is so fundamental for robust tracking, that non-adaptive trackers can be considered outdated and won't be of interest for state-of-the-art performance.

Finally, Kalal et al. differentiate between *generative* and *discriminative* tracking approaches. Generative approaches consider only the appearance of the actual target and can be considered as target matching tasks because the candidate that is most similar in the target representation model is decided as the new target [1]. As a consequence, generative trackers are challenged by cluttered background scenes [18].

Building upon this, discriminative tracking approaches use the background to obtain negative examples. Tracking is then considered as a binary classification task, which represents the decision boundary between the object and the background. The negative background samples in combination with positive samples from the actual target are used to train the target classifier [1].

2.1.1 Introducing the bounding box

As the position of the target object on a prerecorded image sequence is known, it is common practice in the visual object tracking community to simply provide the tracker with the initial position of the object. Like this, the tracker can be initialized with the knowledge about the location of the object, instead of having to deploy some kind of object detection algorithm. This can be done by simply passing the exact coordinates of the object (consisting of values for x, y, width and height) on the first frame to the tracking algorithm, which is also how it is done in the *HIOB* framework.

The coordinates enclosing the target object that is to be tracked are commonly referred to as the bounding box. The bounding box is simply a rectangle containing the object as tightly as possible and is used to derive the vast majority of metrics used to evaluate the performance of a tracking algorithm. The performance measuring metrics will be introduced in section 4.6.

Equipped with this knowledge, a formal definition of the tracking task can be derived:

The tracking task refers to the process of maintaining a bounding box in such a way, that the center distance between the predicted bounding box and the actual bonding box is minimized, while the intersection between the predicted bounding box and the ground truth bounding box is maximized.

In other words, the goal is to update the bounding box in such a way, that the predicted position is as close to the actual position, while the width and height of the bounding box are updated, depending on the width and height of the object's representation in the image sequence.

2.2 Correlation and convolution

The mathematical operation *Convolution* is the fundamental principle used in the *HIOB* framework to track an object over a sequence of images. The mathematical operation *Correlation* is the key component in the algorithm provided by Danelljan et al., which has been implemented and is evaluated as part of this thesis [8, 9]. Thus, a solid understanding of the two operations is necessary.

The two mathematical operations *Correlation* and *Convolution* are, broadly speaking, used to apply one function onto some other function, like applying a smaller image in each location of a bigger image. This can be done in both the spatial or frequency domain. For now, the focus lies on *spatial* correlation and convolution. The spatial domain in this context means that we apply correlation or convolution directly on the pixels values of an image, instead of working with the frequency representation of an image.

The two operations are almost identical, except for one property which causes the operations to have different outputs. What both operations share, is that they both apply a sum-of-products operation between an image f and a kernel w (generally speaking between two functions, but we are only working with images). In the same way, a grayscale image can be represented by a 2D-matrix of values between 0 and 1, a kernel is also just a matrix of values. The size of the kernel defines how much of the image is taken as input at each location, and the coefficients of the kernel determine what the result of the convolution will be [14].

Having a kernel of a specified size already makes an interesting assumption about the input. Applying some kernel at each location of an image versus trying to process the entire input image at once means, that we assume that we can still extract information, even when we don't consider the entirety of the image. For example, we assume that things that are closer to each other are more likely to share some kind of relation compared to things that are further away from each other. This is true for images, where the likelihood of two pixels belonging to the same object is higher when the pixels are closer together [21] compared to when they are further apart. As another example, data-points extracted from a sound wave are more likely to belong to the same word, if they have been extracted closer to each other than further apart from each other.

Those kernels are often also referred to as masks, templates or windows. Now, applying correlation at a specific location (x, y) on an image f, the output $g(x, y)$ at that location is the *sum of products* of the kernel coefficients and the image pixels within the kernel [14]:

$$\begin{aligned} g(x,y) = w(-1,-1)f(x-y,1-y) + w(1-,0)f(x-1,y-0)+ \\ \ldots + w(0,0)f(x,y) + \ldots + w(1,1)f(x+1,y+1) \end{aligned} \tag{2.1}$$

While Equation 2.1 applies a correlation kernel of static size on a static location, Equation 2.2 provides the formula for applying a kernel w of size $m \times n$, where $m = 2a + 1$ and $n = 2b + 1$ at each location (x, y) of an image f, by varying x and y in such a way, that the center of the kernel visits each pixel once, shifting the kernel over the image and producing the operation result. To ensure that the center of the kernel can actually visit each pixel of an image, the image is usually padded by either 1's or 0's. The $\star$ denotes the correlation operation:

$$(w \star f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2.2}$$

Everything described so far applies to both the *Correlation* and *Convolution* operation. The one sole difference between the correlation and convolution operation is, that for convolution, the kernel is rotated by 180° [14]. This is because correlating a kernel w with a *discrete unit impulse*[1] outputs a copy of w, rotate by 180°. This effect occurs, because the last (rightmost, bottom) value of the kernel visits each pixel first, when correlated on the input image f, starting at the top left location.

To counter this effect, the convolution operation simply pre-rotates the kernel by 180°, so that convoluting a kernel with a discrete unit impulse outputs an exact copy of the kernel [14]. Thus, the formula for convolution is very similar to the one for correlation 2.2, with the $*$ denoting convolution:

$$(w * f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{2.3}$$

.

Here, the minus aligns the coordinates of f and w when one function is rotated by 180° [14].

To summarize why it is important to differentiate between Correlation and Convolution: When correlating is used to process a function with a kernel, the output is reversed, while when convolution is used to process a function with a kernel, the output is not reversed (because the kernel has been pre-rotated).

[1] A function (e.g. an image) that contains only one 1 and the rest being zeros, like a black and white image that is just black with 1 white pixel

2.3 The frequency domain an its properties

In the previous section *Correlation* and *Convolution* have been applied in the spatial domain. This section provides an introduction into the frequency domain and shines light on why it is beneficial to instead convolve or correlate two functions in the frequency domain.

It is possible to express a periodic function as the sum of weighted sines and cosines of different frequencies (with the limitation to periodic functions, this is referred to as the Fourier series). A non-periodic function can be expressed as the integral of sines/cosines multiplied by a weighting function[14]. The later case is known as the Fourier transform and is what is of interest for this thesis, as images are not necessarily periodic. The appearance of an image in the frequency domain thus depends on the frequency of its sinusoidal components, where image regions with low variance in the intensity values (e.g. an equally illuminated wall) correspond to sinusoids of low frequency and image regions with high variance in the intensity values (e.g. sharp edges) are represented by sinusoids of high frequency [14].

It is possible to take a function(e.g. an image), convert it into the frequency domain using the Fourier transform, process it while it is in the frequency domain, and then convert it back into the spatial domain by applying the inverse Fourier transform [14]. What makes this interesting is, that the frequency domain (also referred to as Fourier domain) satisfies special mathematical properties especially useful in this context, stating that convolution in the spatial domain is equivalent to multiplication in the frequency domain [14, 5]. This is useful, because applying convolution or correlation in the spatial domain with a $M \times N$ image and a $m \times n$ kernel requires $MNmn$ operations, while it only requires $2MNlog_2MN$ operations to perform and process it in the frequency domain (2 because the functions need to be converted into the frequency domain and back into the spatial domain) [14]. Thus, calculating convolution or correlation in the frequency domain is much cheaper than calculating either operation in the spatial domain.

2.4 Basic correlation filtering

The basic mathematical operations *Correlation* and *Convolution* have been explained in section 2.2. Evaluating Equation 2.2 at each location of the image f, the output (or correlation score) is highest where the image f and the kernel w are similar. That is, because when two high values on the kernel w and the image f align, their product is bigger than when a high value on the kernel aligns with a low value on the image, and vice versa. Therefore, Equation 2.2 finds locations where the values on w match regions on f.

Equation 2.2 has the drawback of being vulnerable to changes in amplitude in either one of the functions w or f. As a result, the *correlation coefficient* is used to perform correlation filtering/template matching, normalizing both functions to amplitude changes [14]:

$$\gamma(x,y) = \frac{\sum_s \sum_t [w(s,t-\bar{w})][f(x+s,y+t) - \bar{f_{xy}}]}{\left\{\sum_s \sum_t [w(s,t-\bar{w})]^2 \sum_s \sum_t [f(x+s,y+t) - \bar{f_{xy}}]^2\right\}^{\frac{1}{2}}} \tag{2.4}$$

Here, $\bar{w}$ is the average value of the kernel w, and $\bar{f_{xy}}$ is the average value of f in the region contained by the kernel w at the current displacement step. The correlation operation described in section 2.2 is used to shift the kernel w over the image f, so that the kernel visits each location in f. The highest possible value obtainable by eq. 2.4 occurs when the normalized w is identical to the normalized region in f.

This process is referred to as correlation filtering or *template matching*, as the kernel can be thought of as a template or sub-image that we try to find in a bigger image. In fact, in this basic case of correlation filtering, that is exactly what is being done.

2.5 Advanced correlation filters

This section introduces two more advanced types of correlation filters, which provide the foundation for the correlation filter that is used in the approach by Danelljan et al., which has been implemented as part of this thesis [8, 9].

2.5.1 ASEF correlation filters

A common way to detect patterns (i.e. locate objects) in images is to correlate an image with an exemplary template of the pattern or object, which is to be located somewhere in the input image [12]. This corresponds to the most basic implementation of correlation filtering. This kind of filtering has many known weaknesses, the most commonly recognized weakness describes the problem of the template only having a high to a near perfect match when the template can be found almost exactly as is in the input image. As soon as the template is only slightly varied on the input image, the response of such a filter becomes unpredictable [4]. There is an entire family of correlation filters that have been developed to try to overcome this weakness that differs mainly in how they construct the filter from the training samples [4]. One of those filters is the *Synthetic Discriminate Functions* (SDF), which respond well to positive images of the target object while suppressing responses to negative training examples [4].

To overcome this and other weaknesses Bolme et al. introduced a new type of correlation filter, called Average of Synthetic Exact Filters (ASEF), that differ from traditional filters in multiple ways: First, where previous filters like SDF only output a single correlation value, ASEF filters are trained using response images, which have a bright peak centered on the target object [4]. As a consequence of completely specifying the correlation output, a complete, exact filter is learned for each training image, which is finally averaged to create the average filter over the entire training set [4].

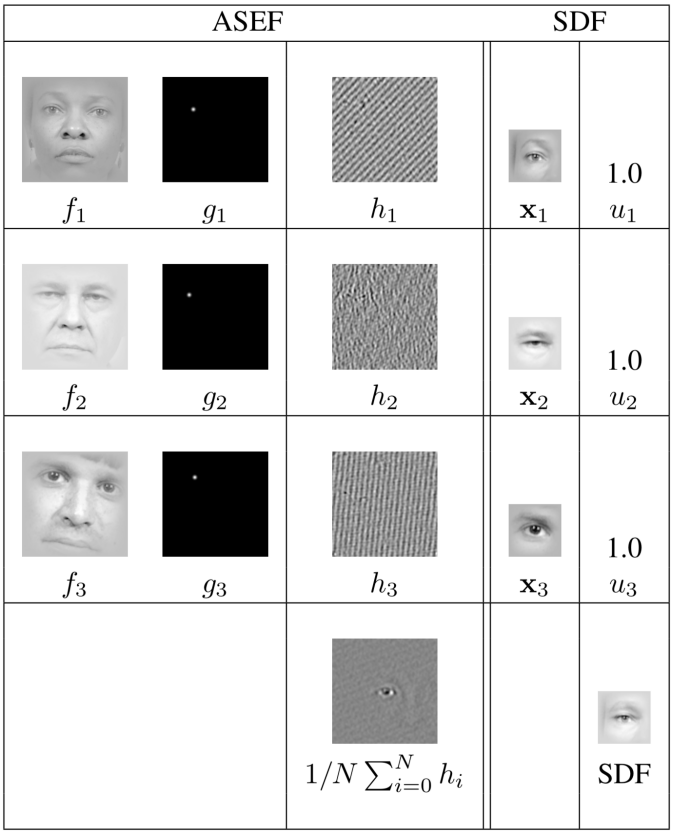


Figure 2.1: Training of an ASEF Filter compared to the training of a SDF Filter. Figure taken from [4].

Figure 2.1 shows the comparison of an ASEF and a SDF filter and will be used to explain the process. A training pair consists of a training image f_i and the desired output g_i. The desired correlation output is synthetically constructed (hence Average of *Synthetic* Exact Filters) and takes the form of a bright peak at the center of the target, here being the left eye. More formally, g_i is defined to be a 2-dimensional Gaussian distribution, centered at the location of the target x_i, y_i with a radius of σ, where the purpose of σ is to find a tradeoff between noise tolerance and peak sharpness:

$$g_i(x, y) = e^{-\frac{(x-x_i)^2+(y-y_i)^2}{\sigma^2}} \tag{2.5}$$

A correlation filter h_i is then computed in the Fourier domain which exactly transforms f_i to g_i. The final correlation is then computed by taking the average of every exact filter.

As stated in the Convolution Theorem and explained in section 2.2, correlation or convolution in the spatial domain is equal to pointwise multiplication in the frequency/Fourier domain [5]. Hence the following equation, where capital letters in the Fourier domain correspond to their non-capital counterpart from the spatial domain holds true, where $\mathcal{F}^{-1}$ denotes the inverse Fourier transform and $\star$ the correlation operation:

$$g(x,y) = (f \star h)(x,y) = \mathcal{F}^{-1}(F(w,v)H(w,v)) \tag{2.6}$$

Equation 2.6 provides the basis for finding an exact filter. In order to solve for the exact filter, first the correlation is computed by including the complex conjugate of H into the equation 2.6:

$$G(w,v) = F(w,v)H^*(w,v) \tag{2.7}$$

Then, the equation can be solved for the exact Filter,

$$H_i^*(w,v) = \frac{G_i(w,v)}{F_i(w,v)} \tag{2.8}$$

meaning that we essentially obtain the exact filter from an element-wise division between the transformed target output G_i and the transformed training image F_i. The final average, exact filter can finally be computed. Because the Fourier Transform is a linear operation, it can be computed in either the spatial or frequency/Fourier Domain, given by the following two equations, where H_μ^* and h_μ are the final ASEF filters:

$$H_\mu^*(w,v) = \frac{1}{N}\sum_i^N =^1 H_i^*(w,v) \tag{2.9}$$

$$h_\mu(x,y) = \frac{1}{N}\sum_i^N =^1 h_i(x,y) \tag{2.10}$$

As it can be seen from figure 2.1, the individual, exact filters h_i do not seem resembles an eye. That is, because each individual filter h_i, that has been produced in the Fourier domain, corresponds to the specific image f_i and exactly transforms this to the output g_i. To produce a filter that generalizes across the entire training set, and thus emphasizes the shared commonalities across the training set, the average of the individual, exact filters needs to be taken, which then takes the shape of an eye. Only the individual filters are essentially weak classifiers that perform perfectly on the corresponding training image but have the same problem mentioned above of being unpredictable for slight variations of the training template.

2.5.2 MOSSE correlation filters

The *Minimum Output Sum of Squared Error* (MOSSE) filter are a newer kind of correlation filter that has been introduced by Bolme et al. [3]. The MOSSE filter is closely related to the ASEF filter described previously. The main drawback of the ASEF approach is, that it requires a large number of training images, which are per definition not available for online tracking tasks, where the tracker is initialized on the first frame of a given video sequence. Therefore the MOSSE filter produces stable and adaptive, ASEF-like filters but instead of requiring a large number of

training images, MOSSE filters can be initialized and trained on a single image [3]. Adaptive in this context means that the filter can be trained online and can adapt to changes in the appearance of an object, which is a key requirement for model representations in visual object tracking.

For training, MOSSE filters require a set of input images f and corresponding training outputs g. Note, that the set can contain only 1 image. A filter h_i that transforms the training image to the output is given by equation 2.11, where the capital letters refer to the non-capital counterparts but in the Fourier domain and the $*$ denotes the conjugate of a complex number:

$$H_i^* = \frac{G_i}{F_i} \tag{2.11}$$

In compliance with most correlation filters, g_i is chosen to be a 2D-Gaussian centered on the target object and the filter is trained in the frequency domain because of cheaper calculation, as described in section 2.3.

In order to find the optimal filter for a set of training images, MOSSE finds a filter H that minimizes the sum of squared errors between the *actual* output of the convolution and the *desired* output of the convolution, which is specified by G_i. The minimization problem that finds the ideal filter is then given by Equation 2.12, where $\odot$ denotes element-wise multiplication:

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{2.12}$$

Without providing the complete derivation (which is provided by Bolme et al.), a closed form expression[2] for the MOSSE filter is:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*} \tag{2.13}$$

Here, the numerator is the correlation between the input and the output, and the denominator is the sum of energy spectra over each image in the training set (this information will be relevant in a second).

Of special interest is the case where only one image is used for training the filter and where small amounts of images are used in the training set. If the training set consists of only one image and the corresponding correlation output, an exact filter is learned, which precisely transforms the one input image to the one correlation output, which can be observed in the following equation:

$$H_i^* = \frac{G_i}{F_i} = \frac{G_i \odot F_i^*}{F_i \odot F_i^*} = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*} \tag{2.14}$$

The problem with exact filters is, that they often fail or behave unpredictably when applied to a new image [3]. In the ASEF approach, this problem has been solved by averaging over the filters from the training set. Just like the ASEF filter, the MOSSE filter produces an exact filter but requires only one training sample

[2] A close form expression is an expression that can be solved with a finite number of operations

in the training set. This becomes clear when a rewritten version of equation 2.9 is considered:

$$H^* = \frac{1}{N} \sum_i \frac{G_i \odot F_i^*}{F_i \odot F_i^*} \tag{2.15}$$

The comparison of equation 2.15 and equation 2.13 shows that in the ASEF case in Equation 2.15, the denominator is much more likely to be close to zero as in the MOSSE case because ASEF utilizes only the energy spectrum of each input image. This is problematic, because when the denominator is close to zero, the filter becomes unstable and the output unpredictable [3]. The ASEF approach tries to counter this problem by averaging over the entire training set, while MOSSE always uses the sum of energy spectra across the entire training set, which is much less likely to produce a small number. Like this, the MOSSE approach is able to produce stable filters even on small training sets.

There is still the problem of MOSSE filters producing an exact filter (which is not desired) when the training set consists of only one image and correlation output. To obtain a bigger training set, n affine transformations[3] are applied on the target object in the initial image, and the desired correlation outputs are generated accordingly [3]. Setting $n = 8$ is enough to produce a stable MOSSE filter, which is not enough for an ASEF filter [3].

Finally, to make the MOSSE filter adaptive, running average in combination with a learning rate η is applied [3] so that the frame i is given by:

$$H_i^* = \frac{A_i}{B_i} \tag{2.16}$$

$$A_i = \eta G_i \odot F_i^* + (1 - \eta) A_{i-1} \tag{2.17}$$

$$A_i = \eta F_i \odot F_i^* + (1 - \eta) B_{i-1} \tag{2.18}$$

To summarize, MOSSE filters can be trained on a single training image (plus the desired correlation output), by generating a small number of training samples using affine transformations on the initial image. MOSSE filters are valid for tracking scenarios, as they can adapt to changes in the appearance of the target object by applying running average and a learning rate on each frame, putting more weight on the recent frames and decaying older frames exponentially over time [3].

2.6 Convolutional neural networks

Convolutional Neural Networks (CNNs) are not a new technology. In fact, they have already been used as early as 1990 [15]. In 1990, LeCun et al. at the AT&T research group developed a Convolutional Neural Network for reading handwritten digits, which was by the end of the 1990s reading 10% percent of all checks in the United States [15, 24]. CNNs became extremely popular after 2012 after Krizhevsky et al. [22] won the ImageNet Visual Recognition Challenge.

[3]Examples for affine transformations are: Translation, scaling or rotation

CNNs exploit the the assumption that values with high proximity have high correlation (more detailed in section 2.2) and don't process the entire input data (e.g. an image) at once, but rather process the input data in small batches, resulting from shifting the kernel over the input, as described in greater detail in section 2.2. By processing the input data in small batches, the CNN can detect small but meaningful visual features like edges [15]. Like this, a CNN does not try to learn *one specific representation* for an object, instead, it learns which collection of detected features correspond to an object.

CNNs make use of the convolution operation in order to process an input image. This gives them major advantages over fully connected architectures. One of those advantages is that the net has to learn fewer weights. This is because at each location only the direct neighborhood needs to be considered (the neighborhood size is determined by the size of the kernel), instead of the entire input data. This property is referred to as *sparse connectivity* [15]. A similar positive property found in CNNs the fact that each parameter (value) in the kernel is used at every position of the input. This is not the case in a traditional neural network, where each weight in the weight matrix is used exactly once in the computation of the output of a layer. This property is referred to as *parameter sharing* [15].

A convolutional layer (or a set of multiple convolutional layers) in a neural network is usually followed by a pooling layer. The job of the pooling layer in a CNN is to remove information that we are not interested in by adjusting the output of the convolutional layers. There are multiple pooling functions, a popular choice is the max pooling function which reports the maximum output within a rectangular neighborhood [15]. In a typical CNN, multiple convolutional layers are stacked after each other, followed by a pooling layer. This set of layers can be repeated, and after enough sets of convolutional and pooling layers, the network architecture is referred to as a *deep* convolutional neural network. Stacking multiple convolutional and pooling layers allows the CNN to learn more and more complex features, as one feature in the nth convolutional layer contains several features from the nth -1 layer.

The output of the final convolution plus pooling set is usually fed into a fully connected layer, for the final calculation of the output.

2.6.1 Tracking with CNNs

As it has been described in section 2.4 the output of the correlation operation (eq. 2.2) is high, when the kernel is similar to the current displacement region on the image. As a result, it is possible to apply the correlation operation to output the position of an object in an image. The same is true for the convolution operation. Taking this idea further, it is also possible to take multiple images that together make up a sequence, and apply convolution on each image and, thus, output a position of an object throughout a sequence. This is the basic idea behind tracking via with CNN's.

In practice, a lot of challenges arise, which is why object tracking is still a challenging problem in the field of computer vision. The most fundamental problem

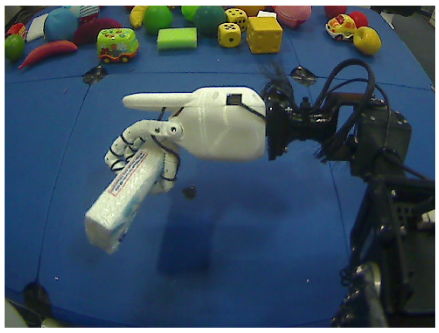
that needs to be considered when we wish to *track* on object throughout a sequence of images, is that objects usually change their appearance through time, be it caused by rotation, changes in illumination and shadows being cast on the object or by a none-rigid transformation of a living thing. This is why a CNN-based tracker usually deploys additional components, dealing with different challenges arising in the tracking context. A tracker can be referred to as CNN-based when a CNN is involved to extract visual features, that are then used to calculate the position of the target object.

A general classification of different CNN-based tracking approaches divides them into two categories [30]: Offline and online trained CNN trackers. Offline and online in the context of neural network training generally refers to the point in time, at which a network has been or is trained. A CNN that has been trained offline (or pre-trained) means the net already learned the weights and biases necessary to accomplish a specific task, like classifying what object is present in an image [22]. This means that the network *already learned* what visual features correspond to an object, and only needs to see (extract) them again in order to know what object is present. Another relevant offline/pre-trained CNN is the VGG16 by Simonyan and Zisserman [29], which is also used in the fully convolutional network based tracker FCNT by Wang et al. [32] which in turn is the basis for the HIOB tracker [30].

The counterpart to offline/pre-trained CNNs are online trained CNNs or networks in general. Online training means, that the CNN learns what features are relevant for a task like tracking during the task itself. On the initial frame of a tracking sequence, the network is provided with the location of an object and, thus, knows that the visual features being extracted from that region correspond to the target object. The following frames are also used for training in order to achieve a more versatile and robust representation of the object. The main two drawbacks of this training strategy are, that the training process is computationally costly, and that as soon as the prediction of the position of the target object is slightly off once, the CNN still assumes that the position is correct and also trains on the features that don't actually belong to the object, reinforcing the error [30]. This can cause the prediction of the network to drift further and further off.

2.7 Histogram of oriented gradients

Histogram of oriented gradients is a feature descriptor, which became widely used after 2005, when the descriptor has been used by Dalal and Triggs on the original MIT pedestrian database, achieving near perfect results [6]. A feature descriptor or visual features in the context of computer vision describes fundamental characteristics of a video or image, like the shape of an object, while filtering out irrelevant noise. HOG is commonly used to derive a feature descriptor for bounding boxes [20]. The basic idea behind HOG is that the appearance of an object can be described well by the „distribution of local intensity gradients or edge directions, even without "[6]. This is implemented by dividing the image (or sub-image) for which HOG features are to be calculated into small cells and calculating a 1-D histogram



(a) The input image showing NICO interaction with the environment.

(b) The HOG feature representation of the input image. The output is produced with a pixel cell size of 16×16, 8 orientations and a block normalization size of 1.

Figure 2.2: The input image frame 417 of the NICO *lift_blue_tissue_01* sequence and the resulting feature representation obtained from the HOG algorithm.

of gradient directions [6], which indicates the orientation of edges in each cell.

To compute HOG for a cell, we first calculate the horizontal and vertical gradients, by filtering (see section 2.2) with the two kernels $g_x = [-1, 0, 1]$ and $g_y = [-1, 0, 1]^T$. Those kernels are applied at each pixel in a cell, for each location the orientation of the resulting gradient can be found by $g = \sqrt{g_x^2 + g_y^2}$ and the magnitude (length) of the gradient can be found by $m = \arctan \frac{g_y}{g_x}$. The magnitude of a gradient of a pixel location is high, when there is a strong change in intensity, according to what has been described in section 2.2 regarding the output of correlation and the similarity between the kernel and the and the section of the image it sees. Thus, those gradients have high magnitude values on edges, and low magnitude values on image regions without high changes in intensity, like on an evenly illuminated wall. This step is referred to as the „Gradient Computation step“ [6].

To obtain a histogram of oriented gradients for a cell, any reasonable number of bins ($n > 9$ is reported to no longer improve performance significantly [6]) is created, by evenly spacing the values between $0° - 180°$ („unsigned “) or $0° - 360°$ („signed “) on the number of bins. It is sufficient to use the unsigned version, as a gradient and the gradients negative are represented by the same number, meaning that a gradient and its 180° opposite are represented by the same number. For the 9 bin examples, the orientation values on the resulting bins in the histogram are $0, 20, 40, 60, ..., 180$.

As the gradient magnitude and orientations have been calculated for each pixel in a cell, each pixel contributes its magnitude score to the bin in the histogram that closest corresponds to its orientation. Those scores are accumulated in the histogram for each pixel in a cell. Like this, the final histogram of a cell has the

(a) Frame 50. (b) Frame 66. (c) Frame 76. (d) Frame 85.

Figure 2.3: The TB100 *Biker* sequence as an example for the challenges of in-plane rotation. Within 25 frames, complete rotation of the object occurred, requiring the tracker to learn a new set of features.

highest values in the bins, whose orientation has the highest accumulated gradient magnitudes. This step of assigning values to the bins in the histogram is referred to as „Spatial binning "[6, 20]. The process of creating a histogram of oriented gradients is repeated for each cell in the image, and the collection of those histograms is referred to as the final feature descriptor.

In order to increase the robustness of the algorithm to illumination and shadowing, contrast-normalization can be applied on the cell histograms. This can be achieved by normalizing the cell histograms in larger spatial groups, referred to as blocks [6]. Normalization can be achieved by dividing each element in the block by the L2 norm of the histogram block vector.

As a closing remark, HOG features refer to one histogram, while a histogram can also be understood as just a vector of values. Thus, a cell feature corresponds to the vector of values in the histogram, and the entire feature representation of the image is the vector resulting from concatenating each cell histogram/vector into on big vector. This is referred to as the final feature vector of the HOG descriptor.

The HOG representation of an image is visualized in Figure 2.2.

2.8 A word on in-plane rotation

Generally speaking, rotation can occur on each axis of an object in 3D space. Each axis poses problems in a tracking context, as the visual features that have been extracted from the image in the spatial domain suddenly no longer correspond to the object, as the features are now differently aligned, or completely new features represent the rotated object. Recalling how a CNN works, the networks learns which visual features correspond to an object. It does not matter where the CNN finds those features (this can be also be considered a weakness of CNNs), as long as they are detected in the input the CNN will have high activation values for that object. Thus, changing features poses an obvious problem for tracking contexts.

There are different approaches for different kinds of rotation, Goodfellow for examples shows how pooling over the output of multiple parameterized convolutional filters can make a CNN invariant to specific rotations, where each rotation has its own filter [15]. A different solution to In-Plane Rotation (IPR) is provided by Du et al., who propose a rotation adaptive correlation filter [11].

When an object rotates around its y-axis enough and it's back is now facing the

camera, within the few frames of the rotation occurring, a tracker has to adapt to the rotation and learn the new features, which represent the backside of an object. Such situations or when its infeasible to maintain different filters specifically for rotated representations of an object are very challenging to deal with and pose a separate, open research topic and are generally not in the scope of this thesis. However, the problem of IPR is related to the central problem of this thesis, scale estimation in visual object tracking. The problems are insofar similar as that strong changes in the scale of an object also cause the learned, visual features to no longer correspond to the object. That is, because a feature in a CNN context technically is tied to a specific region of activations in the network, and this region is different for a representation of the object at a different scale. Note, that the activation region of a feature is independent of the position of the feature on the input image.

The central difference between the two problems is, that with SV, the object is still expected to be present in the input (ignoring extreme cases where the object grew bigger than the viewport or becomes so small that it is no longer detectable at the image resolution), while with IPR, this is not necessarily the case, as the argument can be made that a rotation can change the appearance of an object so drastically, that it no longer looks like the same object. An example for such a transformation is given in Figure 2.3.

Chapter 3

Related work

In this section, the *HIOB* tracking framework will be briefly explained, as the HIOB framework poses the underlying framework for this thesis [30]. Additionally, available approaches to the problem of scale estimation will be introduced and analyzed, in the light of being compatible with the HIOB tracking framework and its usage with the Neuro-Inspired COmpanion (NICO) robot [19].

3.1 The fully convolutional hierarchical object tracker HIOB

The Fully Convolutional Hierarchical Object Tracker HIOB is the tracking framework which poses the foundation of this thesis. The framework has been developed by Peer Springstübe in 2017 and is based on the Fully Convolutional Network based Tracker (FCNT) by Wang et al. [30, 32]. The HIOB tracker consists of multiple components, which handle the different subtasks in each tracking step. This modularization enables easy replacement or reimplementation of the different modules.

The HIOB framework makes use of a pre-trained CNN for feature extraction and employs a second CNN that is trained online, to adapt the model representation to the changing appearance of the object. At initialization, the extracted features are ranked, and the features that have the strongest relevance regarding the specific target object are used to train the online CNN. The online CNN produces a prediction mask, based on which the most likely position of the object is determined. For each following frame, the pre-trained CNN is used to extract features, which are again fed into the online CNN, to update the prediction mask. The new position is found, by generating a set of candidates (i.e. bounding boxes), which are evaluated based on the prediction mask.

To gain a better understanding of the HIOB pipeline, the components and their purposes are briefly explained:

- ROI calculator: This component extracts a *region of interest* around the target object. This is done under the assumption, that the target object can

only travel a limited distance between two frames. Thus, not the entire input image needs to be processed.

- Feature extractor: The feature extractor produces a set of 2D masks based on the region of interest, where one mask indicates the presence or absence of a specific, visual feature.
- Feature selector: The feature selector ranks the extracted visual features based on how likely they are to belong to the target object. The ranking is obtained by another CNN, that is trained to produce the prediction mask based on all visual features.
- Feature consolidator: The feature consolidator trains the consolidator CNN to predict the position of the object with the selected features.
- Pursuer: The pursuer generates and evaluates candidates, from which the best candidates is HIOB's prediction for a frame. The prediction is accompanied by a confidence value, that indicates how good HIOB thinks the prediction is.
- Updater: The updater adapts the model representation of the target object during tracking. Technically, this is done by training the consolidator CNN on the extracted, visual features, if the appearance of the object changed significantly.

Springstübe split the tracking task into four different subtasks, where each subtask contains the components needed to handle the specific subtask. The four subtasks are:

- ROI calculation: Under the assumption that the target object can only translate a limited distance between to frames, the search space for the object in the next frame is limited to the region of interest (ROI), based on the position of the object in the previous frame. The region of interest is often much smaller than the complete input image, which is why the first subtask in the HIOB framework handles the calculation of the region of interest. Like this, only the region of interest needs to be analyzed for each frame, instead of the entire input image.
- Feature extraction: Visual features, that discriminate the object from the background, are extracted using a pre-trained CNN (specifically, the VGG16 from Simonyan and Zisserman [29] is used). The visual features from the CNN are ranked to determine the most significant features and to filter out features that correspond to other objects that the CNN has been pre-trained on.
- Feature Consolidation: The extracted visual features are incorporated into a single feature mask, which can be thought of like a heat-map, indicating the most likely position of the object. This consolidation is done by an online

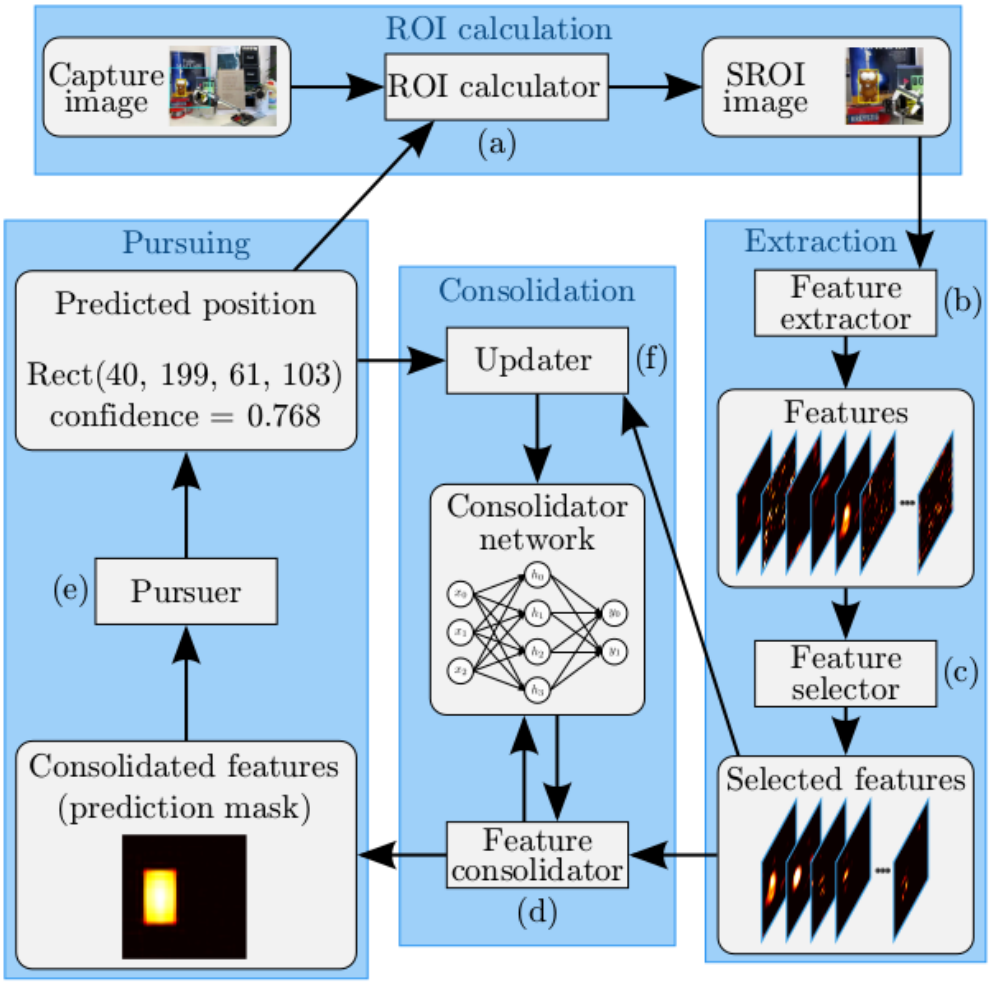


Figure 3.1: The workflow through all components of the HIOB framework. Figure taken from [30]

trained CNN, which updates throughout tracking when the appearance of the object changes.

- Pursuing: The feature mask is evaluated to find the bounding box that corresponds to the most likely position.

The workflow through the HIOB pipeline is visualized in figure 3.1.

Of specific interest is the updater component, which is part of the consolidation subtask. The updaters job is to adapt HIOBs internal object representation to the changing appearance of the object. This can be caused by the object undergoing significant scale changes. There are different strategies used to determine whether an update of HIOBs internal representation is reasonable. If an update is executed, the updated triggers the consolidator network to train on the current frame, so that the predicted position is the output of the selected features. Like this, HIOBs internal object representation now includes a new set of features, corresponding to the changed appearance of the target object.

This is the ideal spot in the pipeline for our scale estimation module to adjust the prediction. Assuming that the scale estimation module accurately outputs the correct size of the object, based on the predicted position, we can now tell HIOB to learn the representation of the object at different scales. This also means that the wrong estimation of the scale would cause the consolidator to learn the wrong features, corrupting the model and resulting in bad tracking performance.

To avoid confusion, the VGG16 feature extractor CNN is fully static, meaning that we can not tell the net to extract different features or scale its features. Thus, the feature extractor remains unaffected from the scale estimation module. However, because of the updater component, we can tell the consolidator CNN (the online CNN) to combine different features to produce the prediction mask. Like this, a representation of the object at the estimated scale is learned.

This concludes the description of the HIOB tracking framework. A full, in-depth description of the framework is provided by Springstübe [30].

3.2 On computational load and real-time performance

In his bachelor Thesis, Tobias Knöppler analyzed the HIOB tracking framework with regard to real-world, robotic applications and both extended and optimized its pipeline to be able to process incoming streams of video data in real time [21]. Building upon this, Heinrich et al. introduced HIOB in combination with the Neuro-Inspired COmpanion (NICO) robot by Kerzel et al. as a research platform for developmental robotic research [17, 19].

This indicates that the employment of HIOB in robotic agents operating on real-time poses a major, practical use case of the HIOB tracking framework, which should be considered in the analysis of available algorithms for the problem of scale estimation.

For specific cases like developmental robotics research, or more generally, real-world tasks, it is obvious that processing speed is a key factor, as robotic agents interacting with the real world need to be able to react to (not only) visual events as fast as possible. For example, autonomous vehicles need to be able to react to the detection of pedestrians or other vehicles immediately and make decisions based on what is happening in the real-world.

This leads to the conclusion that one of the major deciding factors between the available scale estimation algorithms should be the computational complexity of the algorithm, specifying the additional computational load that will be included into the HIOB framework by implementing the algorithm. Under consideration of the above, the focus should be primarily on finding a solid scale estimation algorithm that achieves *strong* results at low computational cost, versus an algorithm that achieves near perfect results but at disproportionally *higher* computational cost.

3.3 Possible algorithms for scale estimation

This section provides an overview of existing approaches to the problem of scale estimation. The approaches are analyzed with the motivation whether they apply to the HIOB framework and its usage with the developmental NICO robot [19].

3.3.1 Using depth-sensors

With the increased availability of cheap depth sensors like the Microsoft Kinect sensor, a novel research field emerged, dubbed RGB-D tracking emerged in recent years. In RGB-D, information about the depth (specifically the z-axis order of object) and the color frame is analyzed to increase tracking results. In classical, RGB tracking, only the color frame is analyzed [1]. RGB-D opens up new ways of tackling classical problems in visual object tracking.

Exploiting the depth information, Meshgi et al. provide a solution to the problem of occlusion of the target object [26]. Additionally, Meshgi et al. handle scale variations by sampling the image with a variety of particles of different size. The particles are selected based on the overlap with the target. However, this is approach requires a scale adaptive model, which Meshgi et al. obtain from exploiting the depth channel.

Camplani et al. extend the RGB correlation filter based KCF tracker to handle scale variations. SE is achieved by computing a set of quantized scale factors (which enables pre-computing specific matrices and speeds up computation) and setting the scale to the factor, that is closest to the depth of the object at the current frame, relative to the depth at the initial frame. The template of the correlation filter is then scaled accordingly.

There are still differences in how exactly the scale is extracted from the depth data. Simply scaling the model based on the available depth data introduces little computational overhead, compared to many other approaches at scale estimation, that often maintain a pool of scale templates or classifiers [8, 9, 25, 34] which makes depth-based approaches specifically attractive for robotic use cases, where it is of high importance that the computation time remains as low as possible so that the robot has time to react to events [21].

While the depth-based approaches seem rather promising, the constraints of the use cases of the HIOB framework have not yet been considered. As mentioned earlier HIOB is used in combination with the Neuro-Inspired COmpanion (NICO) robot, specifically in the fields of developmental robotics research, where the goal is to study and apply human cognitive functions involved in the learning process of children to the fields of robotics. This means, that depth must not be used to derive additional information about the target object during tracking, as young infants still need to develop a sense for depth based stereo vision. This harsh constraint forbids the use of depth-data for scale estimation.

3.3.2 Patch-based

An algorithm capable of detecting and handling scale changes that does not require information from the depth channel has been introduced by Xu et al. [34], in which the central idea is to divide the target object into four different patches and train a classifier on each patch. A classifier, in this case, means that the task of object localization is defined by classifying image locations into either background or object, which is commonly seen and is referred to as tracking by detection [34].

Specifically, a scale factor is calculated between frame t and $t-1$, where p_{t-1} denotes the center position and $w_{t-1} \times h_{t-1}$ denotes the scale in the $t-1$ frame. In the $t-1$-th frame an image patch x^{t-1} is extracted centered on p_{t-1} and resized to be be of $W \times H$. The patch is x^{t-1} is then divided into four patches, whose central locations are $(w_1(t-1), h_1(t-1)), (w_2(t-1), h_2(t-1)), \ldots (w_4(t-1), h_4(t-1))$. On each of those patches a new classifier is trained, so that there a four different classifiers [34]. See figure 3.2 for a visualization of the patch based approach and how it is used to determine the change of scale between two frames.

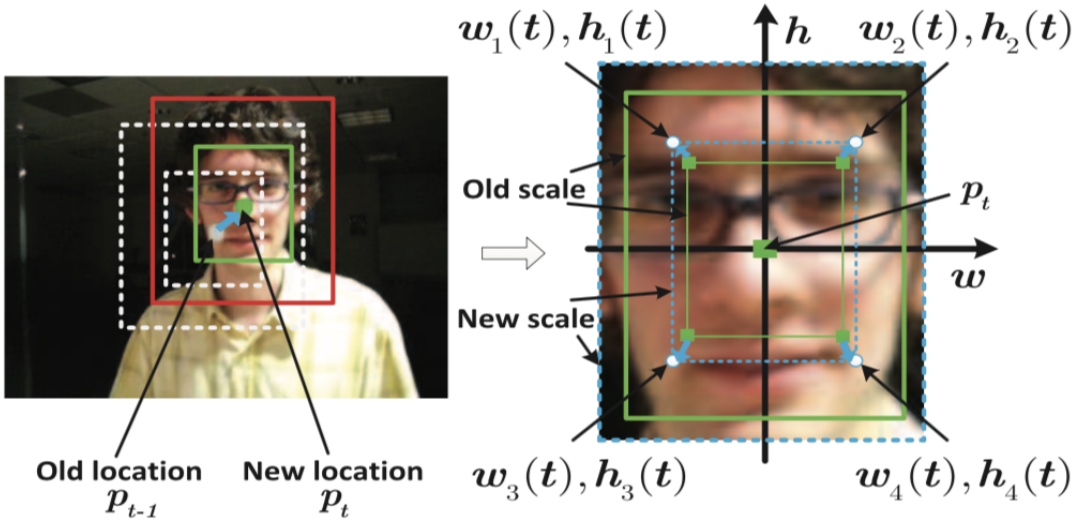


Figure 3.2: Scale calculation using the patch based approach The matching points in the t-th frame are obtained from the classifiers trained on the four patches in the $t-1$-th frame. The new scale is calculated using equation 3.1. Figure taken from [34].

The scale in the t-th frame is obtained after first localizing the object. After localization, an image patch x^t is extracted and resized to $W \times H$ and the four patches are extracted as described above. For each patch a confidence score is calculated, where the highest confidence scores in each patch corresponds to the matching points of the patches in the $t-1$-th and t-th frame, which are denoted by $(w_1(t), h_1(t)), (w_2(t), h_2(t)), \ldots (w_4(t), h_4(t))$. Like this, the scale factor between two frames can be found by the following equation, where δ is a weighting parameter, making the results more robust:

$$\gamma_t = \sqrt{\left(\frac{\sum_{j=1}^{4} \delta_j |w_j(t)|}{\sum_{i=1}^{4} \delta_i |w_i(t-1)|}\right) \cdot \left(\frac{\sum_{j=1}^{4} \delta_j |h_j(t)|}{\sum_{i=1}^{4} \delta_i |h_i(t-1)|}\right)} \tag{3.1}$$

A more detailed described of the patch based classifiers and the model is provided by Xu et al. [34]. While this approach achieves superior results on 10 challenging scale variations sequences from the TB100 dataset [33, 34] it comes at the cost of relatively high computational overhead. In fact, the computational complexity of this approach grows linearly with the number of classifiers deployed [16]. This makes the approach unattractive, as it is a key requirement for robotic use cases to operate in real-time.

3.3.3 Sample-based

This section introduced a collection of algorithms, which follow the general idea of sampling the search space around the target object at different scale factors and finding the sample which best corresponds to the size of the object from the previous frames. Such approaches are intuitive and are widely used in trackers which do not use the depth channel for additional information about the scale. There are different ways of actually determining the best size, like Li et al., who sample their appearance model at multiple resolutions [25].

While the algorithm provided by Li et al. [25] could possibly be integrated into the HIOB pipeline, the approach comes at a large computational cost, as the entire translational model needs to be sampled at multiple resolutions, performing an exhaustive search, to obtain the scale accurately [9]. This makes the approach not suitable for real-time tracking, which is a secondary requirement for an algorithm that could be integrated into the HIOB pipeline.

Contrary to the previous approach, Danelljan et al. [8, 9] propose a 1-dimensional scale filter, that operated independently of the translational model. The scale filter can be applied at different locations and outputs correlation scores in the scale domain, where the highest correlation score corresponds to the ideal scale of the actual object. The fact that the scale filter proposed by Danelljan et al. works independently of the translational model is noteworthy because this means that the scale filter can be separated from the tracker and included into any tracker without a scale estimation component [9].

Closely related to the approach from Danelljan et al. is the approach of Sun et al., who extracted samples from their feature map instead of sampling on the image directly [31]. This could, however, be seen as sampling on the image and then obtaining a feature representation, which is what Danelljan et al. do, emphasizing the similarity of the two approaches.

Du et al. also refer to the work of Danelljan et al. for scale estimation [11]. However, Zhu et al. found that correlating the samples in the Fourier domain is not necessary and instead take the dot product of the differently sized samples in the feature space directly and apply a smooth filter after each estimation [11].

However, the idea of using an isolated scale estimator that operates based on samples at different scale factors is the same.

What can be taken away from this is that while the implementational details of the algorithms differ, the intuitive idea of sampling the image at different scale levels and finding the best fitting sample is shared across the algorithms. The computational complexity of the algorithms is heavily impacted by the specific implementations, and is, thus, hard to generalize. The exhaustive search of the algorithm provided by Li et al. [25], for example, has such high computational cost, that the approach is inapplicable for real-time tracking, while the separate scale filter provided by Danelljan et al. (specifically the *fast* implementation) runs at up to 50 fps, which is fast enough for real-time tracking.

Chapter 4

Approach

The goal of this thesis is to find and implement a robust algorithm for handling scale changes of the target object during tracking. This chapter describes how the goal has been approached. While the available algorithms have already been presented and discussed in Chapter 3, this chapter covers the implementation details and additional design choices. Two main algorithms have been implemented, which both have been extended to be able to output the scale change independently over two axes. Additionally, update strategies have been implemented to further optimize the behavior of the algorithms. A new metric has been introduced for the specific evaluation of the scale estimation algorithm, which will be explained.

4.1 Dealing with the computational load

Including a new module into the HIOB tracker, aiming at correctly estimating the size of the object throughout tracking, comes at the cost of higher computational load. As it has been described in 3.2, it is desirable and necessary to keep the computational load as low as possible, allowing the tracker to process input faster. It has been pointed out by Danelljan et al. [8, 9], that the translational difference of the object between two frames is (usually) greater than the difference of the object in terms of its scale (i.e. the object moves instead of growing or shrinking). Danelljan et al. argue that it is viable to apply any scale estimation algorithm after the tracker already predicted a new position. This reduces the computational load of the algorithm significantly, as the entire algorithm is only applied once per frame, instead of once per possible location. This design choice has been transferred to the HIOB scale estimation module, meaning that independently of the different algorithms that have been implemented (Candidates and DSST), the algorithm will always be executed after the position of the object (on the current) frame has been determined.

4.2 The scaled candidates approach

The first approach that has been implemented is referred to as the *Scaled Candidates Approach*. Note that a *Candidate* simply refers to a set of coordinates (x, y, w, h), describing the target object on the current frame. The broad idea of generating additional, scaled candidates has been introduced by Springstübe in his original work on the HIOB tracker, where the problem of HIOB not being able to handle changes in the size of the target object is brought up [30]. In section 3.1, the pipeline of the HIOB tracker has been examined and explained, including the process of generating a number of translational candidates. The *scaled candidates approach* expands on this, by generating an additional, parameterized number of candidates, which are based on the most likely, predicted position. Similar to the purely translational, unscaled candidates used for determining the position of the object, the candidates for the scale are evaluated on the prediction mask. However, a new set of mechanism and heuristics is included that refines the evaluation process of the scaled candidates, so that accurate estimation of the scale becomes possible.

4.2.1 Generating additional candidates

The process of generating additional candidates to determine the scale of the object at the current frame is rather plain. In order to create those candidates, the unscaled prediction for the position of the object at the current frame is taken as input for the algorithm. Note that this prediction is the position deemed most likely for the position of the object at the current frame, instead of every possible location, for reasons described in the previous section.

The amount of candidates that will be generated is given by the parameter $number\_scales$. To assure that the amount of generated candidates that are smaller and those that are bigger than the input candidate is the same, $number\_scales$ must always be odd. A second parameter $scale\_factor\_between\_patches$ determines the factor which lies between two candidates, determining how finely the scale space is covered. Thus, a high value for $number\_scales$ and a small value for $scale\_factor\_between\_patches$ result in a precise and wide coverage of scale factors, at the cost of higher computational load.

Only varying the width and height of the input candidate is not enough, as the position of the generated candidates is given by the x and y coordinates. If the width of a candidate would be doubled, but the x coordinated left unchanged, the center of the candidate shift towards the right. The same is true for the height and the y coordinate (this is an implementation detail in HIOB). It is, thus, necessary to also adjust the x and y coordinates of the generated candidate in such a way, that the candidate remains centered on the target object. The candidates generated based on the predicted position with adjusted x and y coordinates are visualized in figure 4.1.

Figure 4.1: The additional candidates generated on the 20th frame of the tracking sequence CarScale form the TB100 Dataset. The cyan square marks the region of interest, the yellow rectangle shows the predicted position of the tracker. For a better overview only a few of the generated candidates are shown in magenta

4.2.2 Rating an individual candidate

The process of evaluating the scaled candidates in order to estimate the current scale of the target object is similar to the process of evaluating the positional candidates, which has been described in 3.1. The evaluation processes are similar, as both make heavy use of HIOB's internal prediction mask. For a more detailed description of the feature mask please refer to Peer Springstübes original work [30].

Just like in the evaluation process of the translational candidates, each scaled candidate first gets converted to the coordinate system of the 2D prediction mask. As described in 3.1, the mask is the result of the feature consolidator, which can be visualized as a heat map, with high values corresponding to a high likelihood of the region on the heat map belonging to the target object.

However, in the evaluation of the translational candidates, a score is obtained by summing up the values on the prediction mask that are contained by a candidate. This does not work for the scale estimation problem, because finding the candidate with the highest absolute sum of likelihoods when the scale is varied would mean that the biggest candidates always achieve the highest score. Thus, a mechanism is needed that also punishes candidates, relative to the candidate's size.

Once a candidate has been converted to the size of the prediction mask, the score is calculated that rates each individual, scaled candidate. The score each candidate achieves depends on two threshold parameters, *inner_punish_threshold* and *outer_punish_threshold*. The value of the parameter *inner_punish_threshold* determines, how small the likelihood of each location on the heat map is allowed to be, before the candidate gets punished (punishing the candidate can be thought of as increasing the candidates punishment score) for containing locations with a lower likelihood than the value of that parameter. For each location contained by the candidates that is smaller than the value of *inner_punish_threshold*, the value

of the location on the prediction mask is extracted and saved, when each location has been checked, the saved values from the heat map are summed up, creating the first part of the candidates rating/punishment score.

Similarly, the parameter *outer_punish_threshold* determines the threshold likelihood value at which the candidates rating is reduced for not containing locations that have a higher likelihood of belonging to the target object than the value of the parameter. For each location that is not contained by the scaled candidate that is bigger than the value of the parameter *outer_punish_threshold*, the value of that pixel is extracted from the prediction mask. The sum of all of those values forms the second part of the punishment score for a scaled candidate. The two scores

- Outer punish value: The punishment score of a candidate for not containing values of high likelihood on the predication mask.
- Inner punish value: The punishment score of a candidate for containing values of low likelihood on the prediction mask.

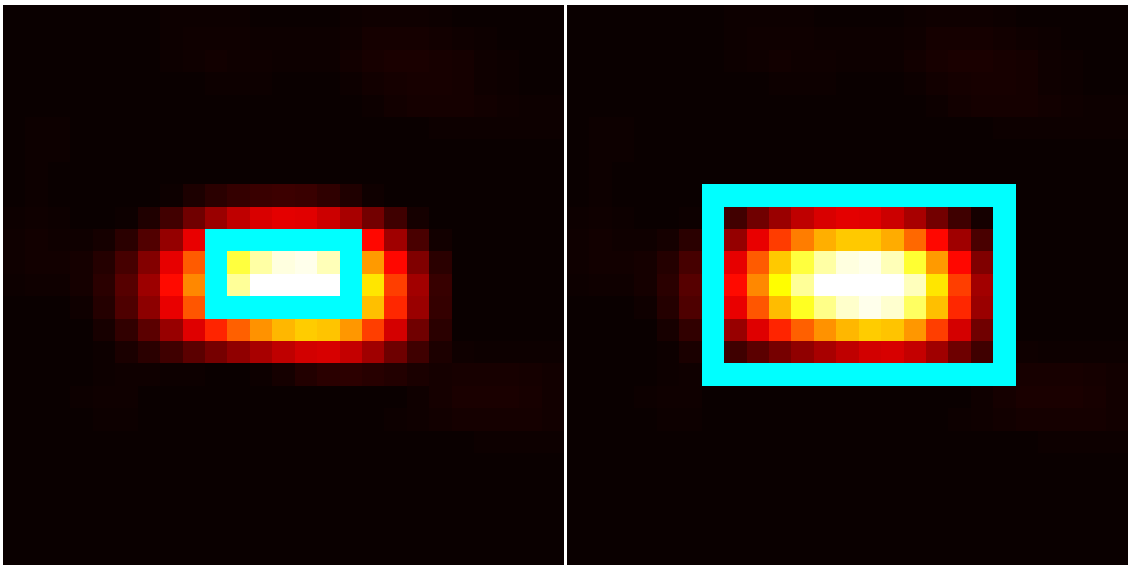
are finally added together to obtain the punishment score of a candidate, which is later used to find the candidate whose size best corresponds to the actual size of the target object on the current frame. This procedure is applied to each previously generated candidate in order to obtain a measurement that can be used to compare and select the candidate with the size closest to the current, actual size of the target object.

To obtain a smoother scale estimation, the parameter *scale_window_step_size* increases the obtained punishment score, depending on how much the scale factor of a candidate divergence from 1. By doing this, the algorithm can only change the scale factor when a candidate achieves a significantly better scale rating and does not produce an unstable size curve. This idea has been borrowed from Danelljan et al, who employ a Hann window to achieve the same effect [9]. Thus, the final punishment score for each candidate is obtained from the multiplication of the *inner outer sum* with the scale window, which increases the punishment for diverging from 1.

The process of finding the punishment score based on the prediction mask for exemplary candidates is shown in 4.2.

4.2.3 Candidate selection

Considering how every candidate, generated by the scaled candidates algorithm, is rated, finding the candidate with the size closest to the actual size of the target object is a trivial task. Selecting the candidate with the size closest to the actual size of the target is done by finding the candidate that has the smallest punishment sum. Recalling how the punishment score of a candidate is calculated, the punishment score corresponds to the sum of likelihoods on the prediction mask, that are either lower than a defined parameterized threshold and contained by the candidates bounding box or higher than another parameterized threshold but not contained by the candidates bounding box and have been multiplied with the scale window.



(a) The smallest generated candidate on the prediction mask, the candidate does not contain locations with small likelihood values, but gets punished for not containing the surrounding values of high liklehood

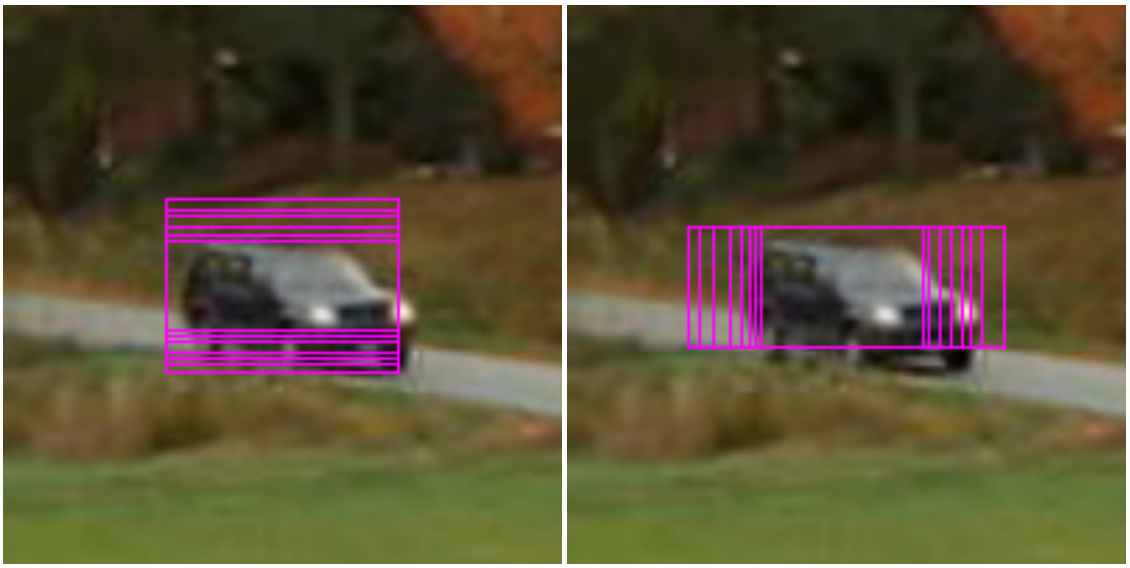
(b) The biggest generated candidate on the prediction mask, the candidate does not get punished for missing values of high liklihood (as oposed to the smallest candidate), but gets punished for containing some values of low liklihood.

Figure 4.2: Two candidates of the same frame visualized on the prediction mask.

After the best fitting candidate has been found, the coordinates of that candidate will be used as the final prediction of the tracker for the current frame (*width* and *height* with adjust x and y values) If the parameter $max\_scale\_difference$ is supplied, the change in size is limited, so that the scale difference compared to the previous frame is not greater than the value of the parameter, this can be understood as a form of regularization.

4.2.4 Candidates: Static aspect ratio

The three preceding sections can be understood as the basic implementation of the Candidates scale estimation algorithm. This idea has been proposed by Springstübe in his original work on the HIOB framework [30]. As this version of the algorithm has already been described in the preceding three sections, it is enough to note that this version of the algorithm is referred to as *static* version of the algorithm, because only one scale factor is maintained. This factor is applied to both the x and y-axis. Thus, the aspect ration remains static during tracking and is determined by the shape of the object on the initial frame.



(a) This set of candidates has been generated by only adjusting the height and accordingly the y coordinate based on the prediction for this frame.

(b) similarly to the set of neighboring candidates, this set of candidates has been generated by only changing the width and adjusting the x coordinate accordingly.

Figure 4.3: The two sets of generated candidate, the 20th frame of the sequence CarScale of the TB100 Dataset has been used. For the sake of better overview, only some wo candidates of the same frame visualized on the prediction mask.of the generated candidates are visualized. The rating of each candidate is achieved as describe in section 4.2.2.

4.2.5 Candidates: Dynamic aspect ratio

This variation of the scaled candidates algorithm is able to handle cases where the object shrinks or grows unevenly over its x and y-axis, compared to the initial implementation of the algorithm, where the same scale factor would be applied on the x and y-axis of the object, assuming that the object never changes its orientation towards the camera.

Such cases, where the scale of the object on the x and y-axis change independently of each other, occur primarily when the real, 3D-object rotates, causing a distortion of the object on the 2D-image representation. As it has been described in section 2.8, the problem of the target object rotating is a different problem with a completely new and different set of challenges and doesn't originally fall in the scope of this thesis. Yet, the problem of estimating the scale of the object and the problem of handling changes in the aspect ratio (i.e. reacting to in-plane rotation of the object) of the object are related, at least in the sense of the tracker having to adapt its internal object representation to match the new appearance of the object. Even though it seems as if an object does not change it's appearance when only increasing or decreasing its scale, this is not true for a convolutional tracker,

where different features visual features represent a bigger or smaller version of an object, thus causing the need to learn a new set of weights.

This relatedness of the two different problems and the fact that only slight changes of the algorithm were necessary to achieve independent scale factors for the x and y-axis lead to the decision of implementing this variation of the static candidate generation algorithm.

The following changes have been made to the static version of the scaled candidates algorithm: Instead of generating only one set of *number_scales* candidates and scaling them accordingly, two distinct sets of candidates are generated. For every set of generated candidates, only one axis (either x or y) is scaled, and the other axis is kept unchanged. This produces a total of $2 * number\_scales$ candidates. There is no need to change how an individual candidate is rated/punished, thus the rating of every candidate is done exactly as described in section 4.2.2.

The distinct sets of candidates that are being generated by only changing one axis of the positional prediction for this frame are visualized in Figure 4.3.

After each individual candidate has been rated, the best candidate from each subset is selected and the scale factors of those two candidates are accepted as scale change on the axis corresponding to the candidate. A final candidate is thus generated, with the scaled x and y-axis according to the best candidates from each set. Like this, it is possible to maintain two independent scale factors, on for the x and another for the y-axis.

4.3 The DSST algorithm

The second algorithm which has been implemented has originally been introduced by Danelljan et al. [8] in 2014. In a second paper by Danelljan et al. [9], the algorithm has been further optimized and enhanced. A reference implementation of the entire *Descriminitive Scale Space Tracker* (DSST) tracker in Matlab is provided by Danelljan et al. [7], from which the components relevant for estimating the scale have been extracted and included in the HIOB tracking framework [30].

The following paragraphs contain a detailed description of the improved version of Danelljan et al.'s algorithm, based on how the algorithm has originally been presented [9, 8]:

Danelljan et al. make use of the Minimum Output Sum of Squared Errors (MOSSE) correlation filters [3], which have been described in section 2.5.2. The DSST tracking framework works by learning two separate MOSSE correlation filters: One 2D correlation filter for estimating the translation (i.e. predicting the position) of the object and a second, 1D correlation filter estimating the scale of the target object. The two-dimensional, translational filter is not relevant for this thesis, because the HIOB tracking framework provides the translational prediction for each frame, so only the part of the algorithm that estimates the scale of the object will be utilized and explained.

The one-dimensional correlation filter can be applied at any location of a frame and compute correlation scores on the scale dimension, where the highest corre-

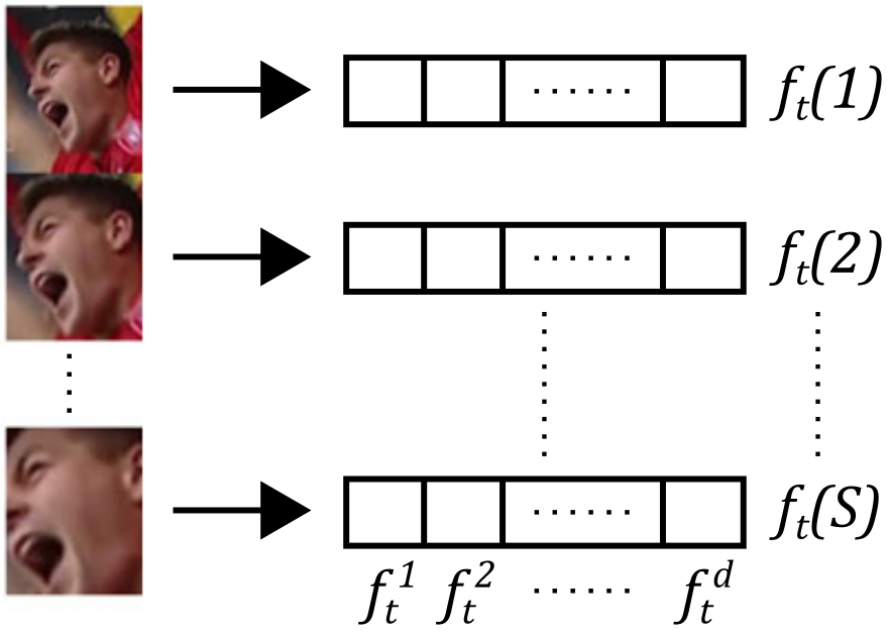


Figure 4.4: The scale sample from the DSST scale estimation algorithm. S scaled patches are extracted from the image at time t. A feature vector f is obtained from the HOG feature descriptor. The length d of a feature vector depends on how HOG is configured but is the same across one sample, because the image patches are resized to the same size. Figure taken from [9].

lation score corresponds to the scale changes of the current frame compared to the previous frame. The one-dimensional correlation filter for the scale dimension works by extracting multiple image patches with varying size, centered on the position of the target object. How many patches are extracted and how fine-grained the scale steps between two patches are, is determined by the parameter values of $number\_scales$ and $scale\_factor\_between\_patches$ (like in the candidate approach, $number\_scales$ is required to be odd).

Each extracted patch gets resized to the initial size of the target object (or to the closest approximation that can be processed by the feature descriptor). Once each patch has been resized to the same size, a d-dimensional feature vector is created by the feature descriptor. More formally, $W \times H$ denotes the initial target *width* and *height* of the object on the initial frame of the tracking sequence, and $number\_scales$ denotes the size of the correlation filter (i.e. how many patches are extracted). For each $n \in \left\{-\frac{n\_scales-1}{2}, ..., \frac{n\_scales-1}{2}\right\}$ an image patch I_n gets extracted. The $scale\_factor$ for that patch is obtained from the $scale\_factor\_between\_patches$ parameter at n steps away from 1. Thus, $scale\_factor_n W \times scale\_factor_n H$, denotes the image patch size at scale level n. The process of extracting multiple patches, which make up a test or training sample for the filter, is visualized in 4.4.

For the d-dimensional feature vector of a resized image patch, Histogram of Oriented Gradients (HOG) features with a 4×4 cell size are extracted. The output

of the HOG algorithm is the d-dimensional feature vector and the collection of each feature vector (one per extracted patch) is referred to as a sample f. The feature domain of the sample is expressed as f^l, where l can be understood as the $i'th$ entry in the feature vector (i.e. one feature), across each extracted image patch I_n in the sample f.

The goal is now to learn a correlation filter h, with one filter h^l per feature (entry in the feature vector) across each image patch I_n in f, which is achieved by minimizing the L^2 error between the actual correlation output and the desired correlation output g. This is described by Equation 4.1, where $\circ$ corresponds to circular correlation.

$$\epsilon = \|g - \sum_{l=1}^{d} h^l \circ f^l\|^2 + \lambda \sum_{l=1}^{d} \|h^l\|^2 \tag{4.1}$$

The second term in (4.1) is a weighted regularization, which is needed to prevent division by zero in the next step. The desired correlation output is typical of Gaussian nature, which is typical for correlation filters.

Eq. (4.1) can efficiently be solved in the Fourier domain, thus, eq.(4.2) is obtained by applying Parseval's theorem [20] to eq. (4.1), where capital letters denote the discrete fourier transformation applied to the specific entity, and the $\overline{G}$ and $\overline{F}$ denotes to the complex conjugate [14] of the specific complex number, for G and F respectively:

$$H^l = \frac{\overline{G}F^l}{\sum_{k=1}^{d} \overline{F^k}F^k + \lambda}, \; l = 1, ..., d \tag{4.2}$$

A detailed derivation of eq. (4.2) is provided by Danelljan et al. [9]. For reasons explained in section 2.3, all multiplications in eq. (4.2) need to be applied pointwise, which can be considered true for other cases in this thesis, except when noted otherwise.

With eq. (4.2), the optimal correlation filter h for a single sample (i.e. one frame) can be obtained. This is insufficient, as multiples samples across time need to be considered to obtain a robust correlation filter h throughout a tracking sequence. The robust correlation filter can be achieved by averaging the correlation error in eq. (4.1) over the samples f_t at different time steps.

As the numerator and denominator of eq. (4.3) can be updated at slightly different steps in the algorithm, the following equations show how the numerator A_t^l and the denominator B_t of the correlation filter H_t^l from eq. (4.2) can be updated with a new sample f_{t+1} from the next time step, with η referring to the learning rate:

$$A_t^l = (1 - \eta)A_{t-1}^l + \eta\overline{G}F_t^l, \; l = 1, ..., d \tag{4.3a}$$

$$B_t = (1 - \eta)B_{t-1} + \eta \sum_{k=1}^{d} \overline{F_t^k}F_t^k \tag{4.3b}$$

The only thing that is left now is to apply the correlation filter. Therefore, in a frame at time t, a sample z_t is extracted. Thus, z_t corresponds to the collection of feature vectors from the extracted image patches at different scale levels. Now Z_t^l can be obtained like described, in the same way as H^l from eq.(4.2) has been obtained from the feature vector for each image patch in eq. (4.1). The discrete Fourier transform of the correlation scores resulting from applying the correlation filter based on the sample Z_t^l are obtained by

$$Y_t = \frac{\sum_{l=1}^{d} \overline{A_{t-1}^l} Z_t^l}{B_{t-1} + \eta} \tag{4.4}$$

,

where A_{t-1}^l and B_{t-1} are the numerator and denominator from the previous frame. The final correlation scores y_t are obtained by taking the inverse discrete Fourier transform $y_t = \mathcal{F}^{-1}\{Y_t\}$. Now, y_t contains a score for each extracted image patch, where the highest score corresponds to the extracted image patch that came closest to the scale of the actual object.

A brief write-up of the DSST Scale Estimation Algorithm is provided in algorithm 1.

Algorithm 1 DSST Scale Estimation Algorithm

Input:
Current Frame I_n
Predicted position p_t
Scale model $A_{t-1,scale}$, $B_{t-1,scale}$
Output: Estimated scale s_t, Updated model $A_{t,scale}$, $B_{t,scale}$

1: Extract scale Sample:
2: **for** $n\_scales, scale\_factors$ **do**
3: Create scaled image patch from I_n, position based on p.
4: Resize image patch to fit to the model.
5: Create feature Vector f^l for current patch.
6: **end for**
7: Get the the main correlation filter Z_t^l based on the collection of feature vectors using eq. 4.2.
8: Get the correlation scores Y_t using eq. 4.4, inserting the correlation filter obtained in the previous step.
9: Find the maximum correlation response in y_t by tacking the inverse DFT of Y_t and backtrack the scale factor that has been used to create the image patch with the maximum response.
10: **return** The scale factor s_t of the maximum correlation score, updates model $A_{t-1,scale}$,$B_{t-1,scale}$

4.3.1 DSST: Dynamic aspect ratio

For the same reasons as described in section 4.2.5, the algorithm by Danelljan et al. [9] has been extended to be able to scale the bounding box independently on its x and y axis. The same approach as described in section 4.2.5 has been used to achieve independent scaling of the x and y-axis.

The main difference to the base implementation of the algorithm is that two sets of image patches are being extracted, each set extracting patches scaled only on either the x or the y axis, leaving the other axis unchanged. After the extraction of the image patches, each patch gets resized to the initial size of the target object, and the feature vector is created, in the same manner as in the basic implementation of the algorithm.

In order to rate the scaled candidates independently of each other, it is necessary to maintain two separate correlation filters, one for each axis/set of scaled image patches. The construction and maintenance of each correlation filter are the same as in the base version of the algorithm. Thus, each correlation filter still outputs correlation scores corresponding to the scaled patch that appears to be closest to the model representation of the target object.

The image patches with the best scale level can thus be found for each axis separately, and the final output bounding box of the algorithm is created by adjusting the predicted position in such a way, that the center of the new bounding box and the center of the input bounding box are the same, while the new bounding box has the new, calculated width and height.

4.4 Update Strategies

As stated in section 3.1, the HIOB tracker is based on the FCNT tracker by Wang et al. [32]. In the initial implementation of the FCNT, Wang et al. introduced, without specifically using the term, some form of update strategies. For examples, Wang et al. update one of their internal neural networks with the most confident tracking result from the past 20 frames, in order to keep the model representation of the target object up to date, which can be understood as a simplistic update strategy. Not running any update strategy would correspond to updating every model or the weights of every net on each frame, making assumptions that might not hold, for example, that the target object is present in each frame of the tracking sequence.

Expanding on the update strategy introduced by Wang et al. [32], Springstübe introduced a number of update strategies that have been included into the HIOB tracker [30]. The same update strategies have been implemented in the scale estimation module. For the experiments that are conducted in chapter 5, HIOB is always configured to use the best performing update strategy. However, it can not be assumed, that the update strategy that works best for the translational prediction also works best for the scale estimation algorithm. Thus, the update strategies described in the following sections only refer to when the Scale Estimation mod-

ule is run, and have no effect on when HIOB is updating its internal model of the target object. By doing this, HIOB still only update its model representation of the target when it is considered necessary, while the performance of the scale estimation module can be explored independently of HIOBs configuration.

4.4.1 The max update strategy

This update strategy is naive, running the scale estimator on each frame and provides a baseline for comparison. The update strategy *Max update* (short max) thus utilizes each frame as training date, not making any assumption about the goodness of the current frame. For the sake of keeping the scale stable and trying to achieve a smooth scale curve instead of jumps in the scale caused by bad frames, the parameter *limit_scale_change* is supplied for the Candidates algorithm, with a value of 0.1, which has been determined in the the parameter optimization to achieve the best results for this strategy. While different operations can be used to achieve a smoother scale output over time, many scale estimation algorithms employ a mechanic like this [9, 11].

4.4.2 The confidence window strategy

The update strategy *High Gain Combined* (HGC) is the most promising update strategy introduced by Springstübe [17] for the translational prediction. The update strategy tries to keep the representation of the target object updated, by enforcing the execution of the scale estimation module at least once every 20 frames. Additionally, the model is only updated, when the confidence of the final, translational prediction is greater than 0.2 and less than 0.4. This confidence window aims at only updating the model when the appearance of the target changed significantly (confidence must be less than 0.4), but also prevents bad frames (e.g. strong occlusion) from corrupting the model (confidence must be greater than 0.2).

This idea is adopted for the scale estimation module. In order to only update the scale when the object actually changed its scale, a frame is only considered to have a significant scale change when the confidence of the frame is less than 0.4. To ensure that the scale estimation model is not executed on a bad frame, the same lower confidence threshold of 0.2 must also be satisfied for the scale estimation module to execute. If the "20 frames without execution" mark is exceeded, updating the scale and the is triggered, independently of the confidence of the current frame.

To prevent confusion between the update strategy for HIOBs consolidator CNN and the scale estimation module, the update strategy that controls the scale estimation module will be referred to as the *confidence window strategy* (CWS), even though it is an implementation of Springstübes HGC update strategy.

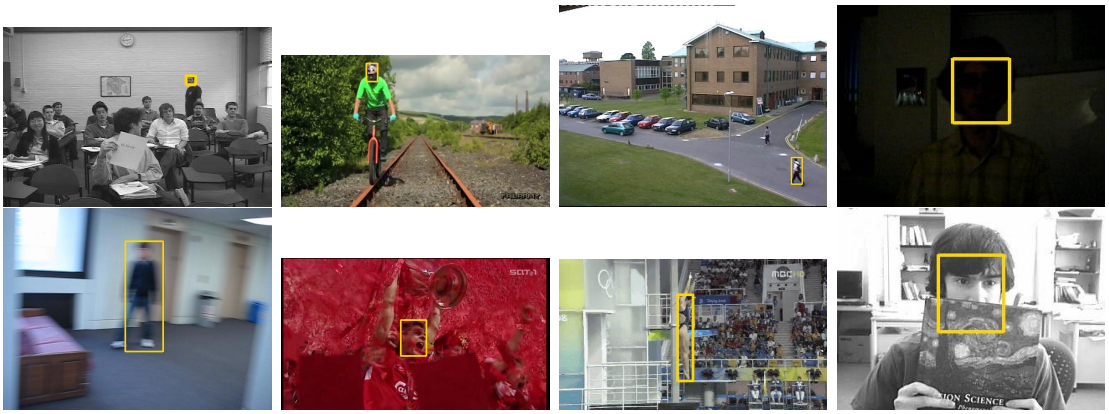


Figure 4.5: Exemplary frames from the TB100 dataset, which contains diverse footage from different settings. The datset shows multiple challenging aspects, like cluttered background, scale variations, low resolution footage, motion blur, occlusion or illumination variations.

4.5 Datasets

Two distinct tracking datasets and a small training subset have been used in this thesis.

The TB100 dataset by Wu et al. is the first dataset used in this thesis. The Online Tracking Benchmark has been created with the specific goal of satisfying a diverse range of challenging tracking conditions [33]. Initially, Wu et al.'s dataset consisted of only 50 sequences, but the dataset has been extended later. The more recent, extended version of 98 distinct sequences (two sequences contain two targets and are treated like two different sequences) will be used in this thesis and will be referred to as TB100 dataset.

The sequences in the TB100 dataset are annotated with attributes describing the main challenges in the different sequences. By finding the sequences that share a specific attribute, a subset can be found corresponding to that attribute. For example, all sequences that have the Scale Variation (SV) attribute are selected into the SV-subset. This enables in-depth analysis and comparison of different trackers, as conclusions can be drawn regarding the strengths and weaknesses of a tracker for the specific attribute, like the performance on sequences with the attribute scale variation or In-Plane-Rotation (IPR). It should be noted, that the attributes are not unique per sequence, meaning that one sequence usually has multiple attributes and in turn is part of different attribute-subsets. An exemplary overview of sequences from the TB100 dataset is provided in Figure 4.5.

Motivated by this, the tracking results of the different algorithms in this thesis will be reported by finding the average achieved metrics scores over the attribute subsets. This summarizes the results for each tracking sequences into broader groups, while insight responding to the attributes of the different groups can be gained.

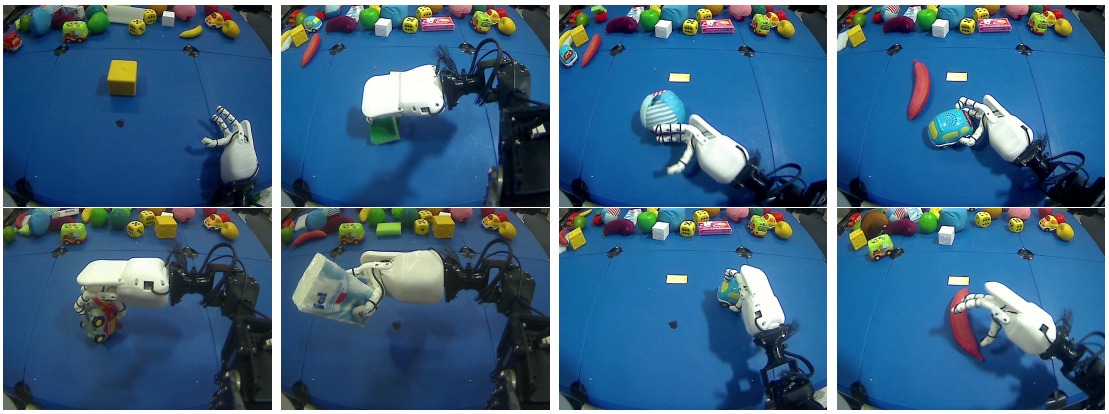


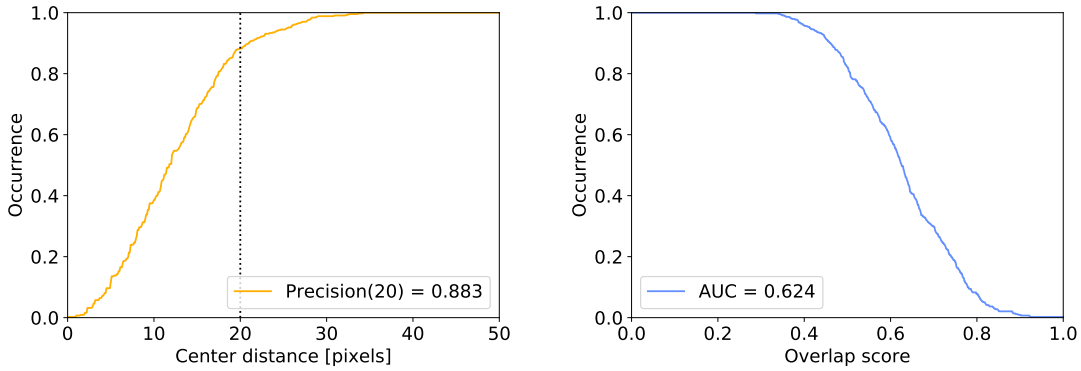
Figure 4.6: Exemplary frames from the NICO dataset, which contains HD-RGB footage of the NICO robot interacting with various objects. Typical challenges for robot-environment interaction, like strong occlusion of the target by the robot's hands, are present in the entire dataset.

A small training subset has been formed from the TB100, consisting of 20 sequences. The sequences have been selected with regard to the attribute balance in the subset, so that the percentage-wise occurrence of each attribute over the sequences is as equal as possible, partly limited by the fact that some attributes have few sequences in the complete TB100, to begin with.

The NICO dataset is the second dataset that is used in this thesis and has been created by Tobias Knöppler [21]. Contrary to the TB100 dataset, which tries to cover a wide spectrum of sequence categories, the NICO dataset consists of 60 sequences that have exclusively been filmed through one of the onboard eye-cameras of the NICO (Neuro-Inspired COmpanion) robot. Thus, the sequences show the NICO robot interacting with various objects, introducing typical challenges for robot-environment interaction like occlusion of the target object by the robots hands [21]. Exemplary frames from the NICO dataset are provided in Figure 4.6.

One of the practical use-cases for the HIOB tracking framework is to employ HIOB in developmental robots like NICO [17]. For this reason, the performance of the different approaches on the NICO dataset should be thoughtfully analyzed, as the NICO dataset was designed to specifically capture robot-environment interactions.

On a technical level, each sequence consists of a set of images, bounding box coordinates of the object for each image and optionally a set of attributes for each sequence. This is the case for both the TB100 and the NICO dataset. Based on the coordinates, the predictions of the tracker can be compared to the ground truth and the performance of the tracker can be evaluated.



(a) The precision plot with the corresponding precision score. The vertical line indicates the threshold value of 20px. The precision score of 0.883 means that 88.3% of all frames have a center location error smaller than 20px. For datasets recorded at varying resolutions, the threshold value of 20px needs to be adjusted.

(b) The success plot with the corresponding success score. The plot shows the success score between 0 and 1. The rating score is the area under the curve.

Figure 4.7: The example plots for the metrics precision and success in comparison.

4.6 Evaluation metrics

A total of three different metrics is used to evaluate the tracking results in this thesis. Additionally, the time needed for the different algorithms will be gathered, in order to determine whether an algorithm is feasible for real-time application on the NICO robot [21]. The first two metrics, Precision and Success are used from the Visual Tracker Benchmark by Wu et al., who also provide the TB100 Dataset[1] [33]. Those two metrics are widely used throughout the visual tracking community, and can thus be used to compare the results of the HIOB tracker, specifically with the scale estimation module, with other trackers. A third metric will be used to specifically measure the results of the scale estimation module, which will be referred to as *Size error*.

4.6.1 The precision plot

The precision plot has initially been introduced by Babenko et al., but is also used by Wu et al. to report the results of their tracking analysis[2, 33]. Previously, tracking results were often reported by taking the frame number and the center location error (the Euclidean distance between the center of the predicted bounding box and the center of the ground truth bounding box). The mean values over the number of frames would then be taken as a measurement of tracker performance.

[1] `http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html`

As pointed out by Babenko et al. [2], this mean of the center location error could be distorted, if the tracker predicted a position close to the ground truth for the majority of the tracking sequence, but completely lost the object on the last few frames and outputted random values, driving the mean center location error up and skewing the data.

Motivated by the above, Babenko et al. introduced the Precision Plot, which shows the percentage of frames, for which the predicted object position is within a distance threshold to the ground truth position. The distance threshold is set to 20 pixels and in order to obtain a numerical score for the precision rating of a tracker, the value on the curve at the threshold number is reported.

An exemplary Precision Plot with further instructions on how to read those plots is provided in figure 4.8a.

4.6.2 The success plot

The success plot as a measurement of tracking performance has been introduced by Wu et al. [33] in their benchmark. The metric measures the overlap between the bounding box predicted by the tracker and the ground truth bounding box. More formally, given the tracking bounding box bb_{tr} and the ground truth bounding box bb_{gt}, the overlap score is defined as $S = \frac{|bb_{tr} \cap bb_{gt}|}{|bb_{tr} \cup bb_{gt}|}$, where $\cap$ and $\cup$ refer to the intersection and the union of the region, while $|\odot|$ denotes the number of pixels in that region. The overlap score is then plotted versus its occurrence. To obtain a numerical rating for the success score of a tracker on a tracking sequence, the area under the curve (AUC) is reported.

An exemplary Success Plot with further instructions on how to read those plots is provided in figure 4.8b.

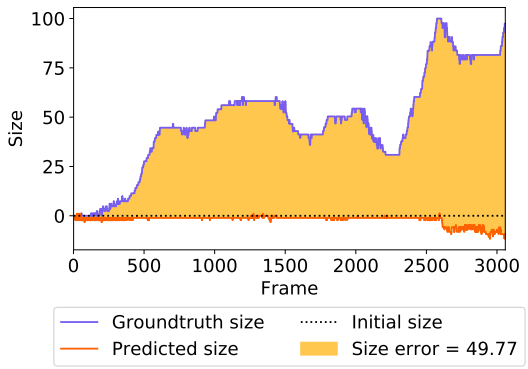
4.6.3 The size error

As a final metric, specifically to measure the quality of the results of the scale estimation module, the size error is introduced in this thesis and will be used excessively to compare the results of the different algorithms. The size error simply refers to the area between curves, where the two curves are the size of the bounding box predicted by the tracker ($size = width \times height$) and the size of the ground truth bounding box.

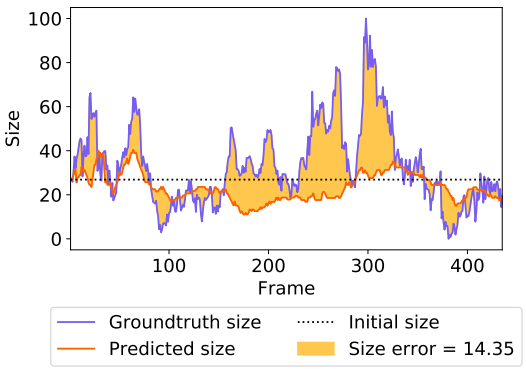
In order to obtain representative values, independent of the size of the bounding boxes, the two curves are first normalized between 0 and 100, based on the scale change in the ground truth. This needs to be kept in mind, because when sequence have very little (or none) scale variation in the ground truth, due to the normalization, the size error in the prediction can look as if it is drastically high, when in fact the difference between the two curves is very small.

After calculating the area between the two normalized size curves, the value is divided by the number of frames in the sequence, which functions as a way of averaging over the sequence length, which ensures comparable results, independently of the length of a tracking sequence.

An exemplary size error plot with further instructions on how to read those plots is provided in figure 4.8.



(a) The size error plot of a sequence with unsuccessfull scale estimation. The size error is high, because the area between the ground truth size curve and the predicted size curve is high.



(b) The size error plot of a sequence with strong scale estimation results. The size error is low, because the area between the ground truth size curve and the predicted size curve is low.

Figure 4.8: Two examplary size plots. The orange line shows the size predicted by the tracker. The indigo line shows the ground truth size. The horizontal, dotted line indicates the starting size of the object at the initial frame. The area between the curves is in yellow. The area between the curves is reported as size error.

Chapter 5

Evaluation and analysis

To measure the performance of the individual approaches described and implemented, a set of experiments has been conducted. Firstly, a parameter optimization has been executed based on the TB100 training subset. Based on the results from the optimization, the ideal tracking configurations have been used to validate the implementation of the DSST algorithm against the reference implementation. Additionally, the different algorithms have been tested on the complete TB100 and NICO dataset, while a final analysis has been conducted exploring the quality of the results with consideration of realistic constraints.

5.1 Parameter optimization

A parameter optimization experiment has been conducted, with the goal of finding the best performing parameter configuration for each approach. As the implementation of the *dynamic* versions of the candidates and DSST algorithms essentially applies the *static* versions of the algorithms twice (once one the x and y axis), the simplifying assumption has been made that ideal parameter settings for the dynamic versions of the algorithms are the same as the ideal parameter settings for the static versions. The scale estimation module has been configured to use the Max update strategy because this is the most basic update strategy.

For each approach, a baseline parameter configuration has been defined, which remains the same across the optimization runs, except for the one parameter that is optimized at a time. Like this, the effect of only changing one parameter value from the baseline configuration can be observed in the results of the parameter optimization. In order to find the best performing set of parameter values for each algorithm, a set of values has been generated for each individual parameter in each approach. The set of values for a parameter has been generated by taking the value of the parameter in the baseline configuration and defining a step function that exponentially modifies the base value. This covers the space close to the base value in detail, while values further away from the base value are explored at a greater interval.

To even out the fact that the predicted position for each frame is not determin-

istic in the HIOB framework, the entire parameter optimization has been executed twice, and the average values are reported. This has been deemed necessary because the DSST algorithm is strongly dependent on the predicted position, which is used to extract samples in the scale domain.

Finally, the best performing values for each parameter are selected to obtain the final configuration for each specific approach. The focus during the selection of the best parameter values lay primarily on the success metric, while the frame rate of the parameter values has also been considered, which measures the computational load of a specific parameter value. However, there were no cases where the frame rate was the deciding factor because whenever the frame rate changed significantly depending on the parameter value, the parameter value that maximized the success metric was in accordance with a high frame rate. A good example of this is the optimization of the DSST parameter *Hog cell size*, which can be inspected in Figure 5.1d, where the best parameter value (as measured by success) does not have the highest frame rate but is very close to the parameter values that reaches the highest frame rate.

To avoid confusion about the two frame rates that are reported: The general frame rate measures the overall frame rate of the entire HIOB tracking framework (which is the sum of all frame rates of the isolated modules involved in the tracking process). The second SE frame rate specifically measures at which frame rate the isolated SE module is running. Thus, the SE module only contributes a small part to the combined computational load of all the modules involved in the tracking process, which explains why a high SE frame rate does not correlate with a high overall frame rate.

The parameter optimization has been executed on the TB100 training subset, on a machine with the following hardware components: 32 GiB RAM, Intel Core i7-4930K at 3.40GHz CPU, two Geforce GTX 1080 GPUs with 8112 MiB video RAM.

5.1.1 DSST optimization: Parameter settings

With an exception to the *hog_cell_size* parameter, the values from the paper by Danelljan et al. have been used for the baseline parameter configuration [9], from which one parameter is varied during the optimization. The following parameters have been optimized:

- *Number scales*: This parameter controls the number of samples that are extracted at different scale levels. The baseline parameter value used in the optimization of the other parameters was set to 33.

- *Scale sigma*: This parameter controls the standard deviation (i.e. the sharpness) of the desired Gaussian correlation filter output, depending on the *Number scales* parameter value. The baseline parameter value used in the optimization of the other parameters was set to 0.25.

- *Learning rate*: This parameter controls how quickly the impact of old frames on the scale model decays over time. The baseline parameter value used in the optimization of the other parameters was set to 0.025.
- *Hog cell size*: This parameter controls the size of each cell in the hog feature extraction algorithm. The block normalization size is adapted to always normalize over two cells, which is not displayed in the hog optimization plot. The baseline parameter value used in the optimization of the other parameters was set to 1×1, which is the only value not in accordance with the original settings by Danelljan et al, were a cell size of 4×4 was used. [9].
- *Scale factor*: This parameter controls how fine-grained the scale space is covered by setting the scale difference between the extracted patches on each frame. The baseline parameter value used in the optimization of the other parameters is 1.02, which is the value is that is also used in Danelljan et al.'s original paper.
- *Scale model max size*: This parameter limits the pixel size an object is initially allowed to have, the value refers to the product of the x and y axis. If an object is bigger than the allowed size on the initial frame, a new size is calculated preserving the aspect ratio of the object, to which the object is then resized. The object representation on the following frames are then always adjusted by the same factor. The baseline parameter value used in the optimization of the other parameters is 512, which is the value is that is also used in Danelljan et al.'s original paper.

5.1.2 DSST optimization: Results

Figure 5.1 show the results obtained from the optimization of the parameter optimization of the DSST algorithm. The following values for the different parameters achieved the best results, measured by the success metric.

- Number scales: 17
- Scale sigma factor: 0.75
- Learning rate: 0.01
- Hog cell size: 4×4
- Scale factor between patches: 1.01
- Scale model max size: 373

The most impactful parameter value is the value of the *number_scales* parameter. Extracting only 17 scale patches with a factor of 1.01 between them means, that only 8% of the scale space around the object at scale 1 is covered. In other words, the smallest extracted patch has been scaled by a factor of 0.92, while the

biggest extract patch has been scaled by a factor of 1.08 when 17 patches are extracted. This is a big difference to having 33 scale samples with a factor of 1.02 between them, as in this case 32% of the scale space around the object at scale 1 is covered. While it is very unlikely that any object recorded at a smooth frame rate would shrink or increase its size by 32% between two frames, a value of 17 for the number of scale samples is alarming for a different reason. Each scale sample consists of a number of extracted and resized images patches, as described in section 4.3. The feature vector of each extracted patch then gets multiplied by a Hann window, which punishes extracted samples more, that are further away from the center. In other words, images patches that are extracted at scale levels that diverge further from 1 are more and more punished, where the image patches extracted at the outmost scale factors are reduced to zero. This is done to stabilize the output of the correlation filter [3].

The stabilizing Hann window[1] is not of static size, it also depends on the number of scale samples that the DSST algorithm extracts, so that the utmost scale factor patches are always reduced to zero. This means that when there are fewer samples, to begin with, the punishment curve is a lot steeper. To give an extreme example, if the number of samples parameter were to be set to 5, there would only be two smaller and two bigger scale samples. The multiplication with the Hann window would punish the outmost samples to be exactly zero, the middle samples to exactly the half of their original score, while the sample in the middle (at scale factor 1), would not be punished at all.

When 17 image patches are extracted, the Hann window is stretched over 8 patches in each direction, but still, the punishing effect on even the samples directly neighboring the patch at scale factor 1 is drastic. This punishment in combination with a scale factor between the patches of only 1.01 (which means that the features of direct neighbors are likely to not be vastly different) has such a strong impact, that the DSST algorithm always predicts a scale factor of one, i.e. never changes the size. Said differently, the parameter optimization produced a configuration, that is equivalent to not running the scale estimation algorithm at all because it forces the algorithm to always produce a scale factor of one.

To eliminate the possibility that this is behavior only present in the HIOB DSST implementation, the parameter configuration has been tested in the DSST reference implementation provided by Danelljan et al., where the same phenomenon has been observed.

While there are multiple conclusions that could be drawn from this finding, it is likely that different ideal parameter values would have been obtained if the different parameters would not have been optimized independently from one another. A different conclusion would be that while the algorithm can achieve good scale estimation results, it also introduces too much inconsistency, cause overall worse average. However, a final evaluation and analysis of the algorithm is due later in

[1] You can think of a Hann window like a bell curve, with its maximum point at one, so that multiplication with a Hann window leaves the center value unchanged, but reduces the outer values.

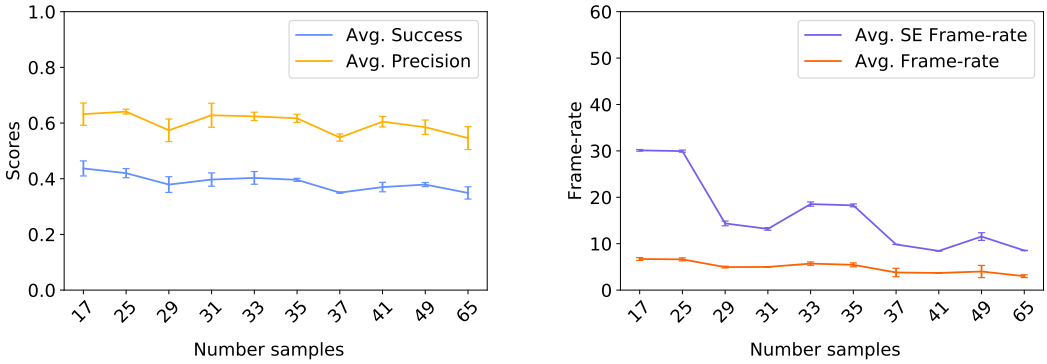
this chapter and will be postponed for now.

As a consequence of the discovery that the ideal parameter configuration causes the DSST algorithm to never change the scale, the parameter configuration is discarded and the values proposed by Danelljan et al. in the original paper are used for the following experiments. This is done because there are now meaningful conclusions that could be drawn regarding the DSST algorithm if it were to be used but could never actually change the scale factor of the object during tracking.

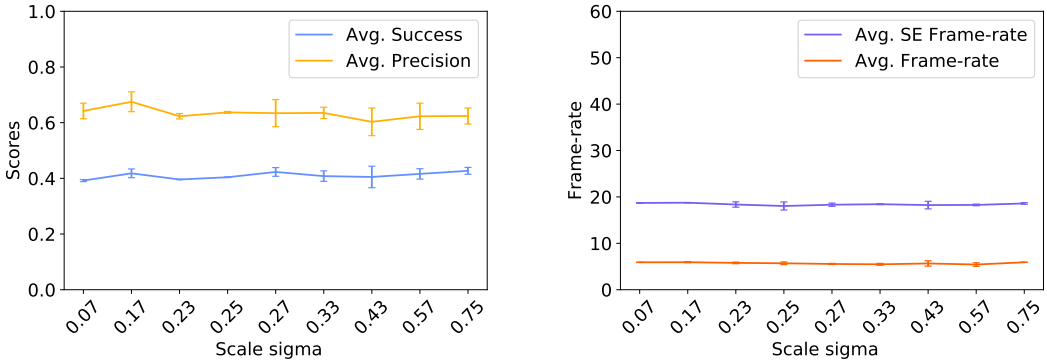
5.1.3 Scaled candidates: Parameter settings

For the optimization of the Scale candidates algorithm, a baseline configuration has been used with relatively neutral parameter settings. The following parameters were optimized, note that the Max update strategy and the *static* version of the algorithm were used for this algorithm, as in the DSST optimization:

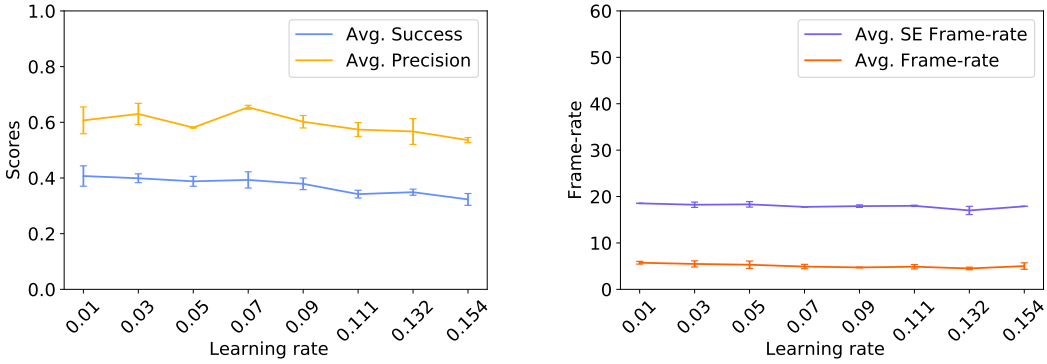
- *Number scales*: This parameter controls the number of candidates that are generated at different scale levels. The baseline parameter value used in the optimization of the other parameters is 33.
- *Scale factor between candidates*: This parameter controls how fine-grained the scale space is covered by setting the scale difference between the generated candidates on one frame. The baseline parameter value used in the optimization of the other parameters is 1.01.
- *Inner punish threshold*: This parameter controls when candidates are punished for containing bad likelihood values for pixels belonging to the object. The baseline parameter value used in the optimization of the other parameters is 0.5.
- *Outer punish threshold*: This parameter controls when candidates are punished for not containing good likelihood values for pixels belonging to the object. The baseline parameter value used in the optimization of the other parameters is 0.5.
- *Max scale difference*: This parameter controls how much the scale is allowed to change between two frames. If the scale change is bigger than allowed by the parameter, the candidates is selected that is closest to the allowed scale difference. The baseline parameter value used in the optimization of the other parameters is 0.01.
- *Scale window step size*: This parameter controls how much the rating of a candidate is punished based on its divergence to the unchanged scale. Thus, candidates generated at the outer extreme scale levels are punished by a lot, while candidates generated close to the original scale are only slightly punished. The baseline parameter value used in the optimization of the other parameters is 0.005.



(a) The average metric scores and the average impact on the computational load of the values of the *number_samples* parameter. The best performing value is 17.0, with an average success rating of 0.437 and an overall tracking frame rate of 6.699.

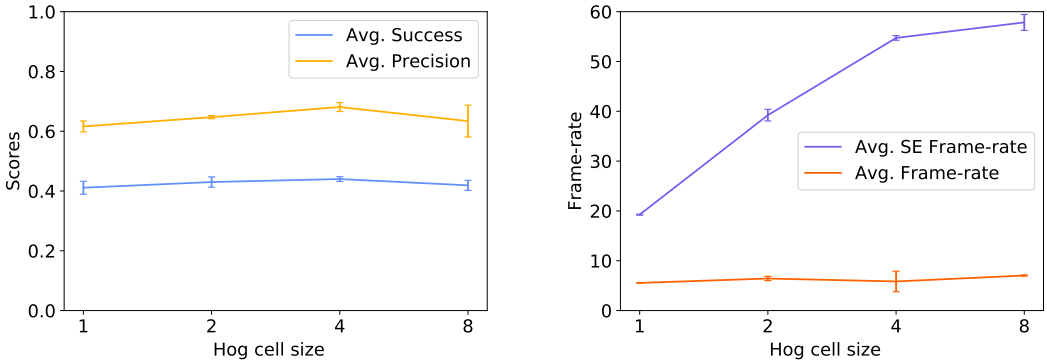


(b) The average metric scores and the average impact on the computational load of the values of the *scale_sigma* parameter. The best performing value is 0.75, with an average success rating of 0.427 and an overall tracking frame rate of 5.91.

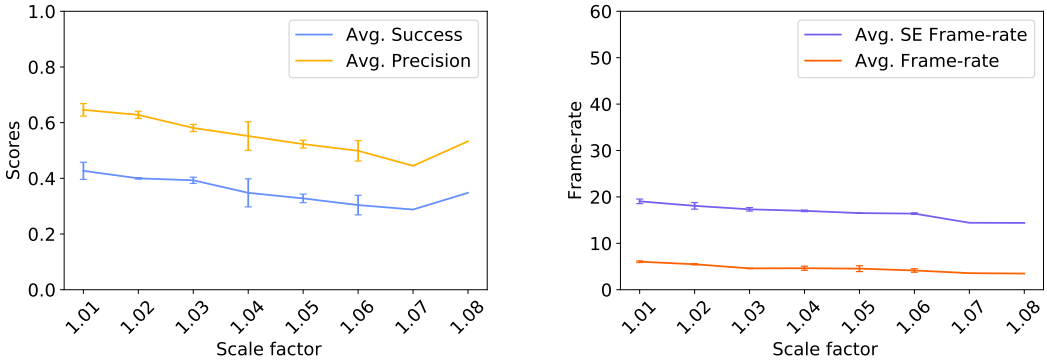


(c) The average metric scores and the average impact on the computational load of the values of the *learning_rate* parameter. The best performing value is 0.01, with an average success rating of 0.407 and an overall tracking frame rate of 5.735.

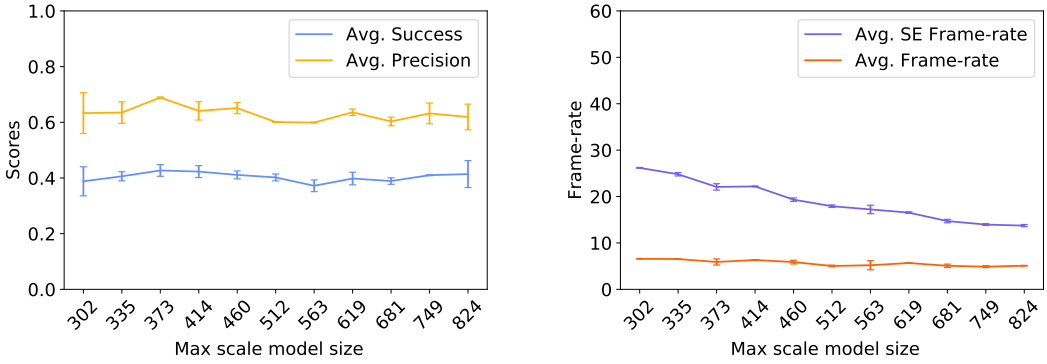
Figure 5.1: Results from the DSST parameter optimization.



(d) The average metric scores and the average impact on the computational load of the values of the *hog_cell_size* parameter. The best performing value is 4.0, with an average success rating of 0.44 and an overall tracking frame rate of 5.847.

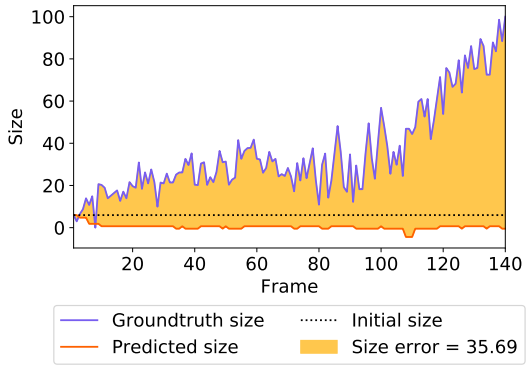


(e) The average metric scores and the average impact on the computational load of the values of the *scale_factor* parameter. The best performing value is 1.01, with an average success rating of 0.427 and an overall tracking frame rate of 6.042.

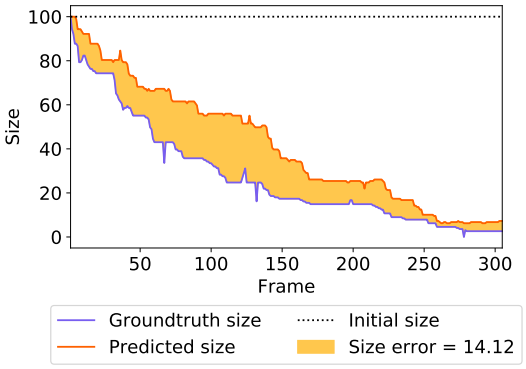


(f) The average metric scores and the average impact on the computational load of the values of the *scale_model_max* parameter. The best performing value is 373, with an average success rating of 0.427 and an overall tracking frame rate of 5.899.

Figure 5.1: Results from the DSST parameter optimization (continued).



(a) The plotted size score of the candidates static approach on the TB100 sequence *Couple*.



(b) The plotted size score of the candidates static approach on the TB100 sequence *Human9*.

Figure 5.2: Comparision between sequences where the object shrinks and where the object grows, using the parameter configuration obtained from the optimization.

5.1.4 Scaled candidates optimization: Results

The results of the parameter optimization of the static candidates algorithm produce the following configuration:

- Number scales: 17
- Scale factor between patches: 1.01
- Inner punish threshold: 0.4
- Outer punish threshold: 0.45
- Max scale difference:
- Scale window step size: 0.2

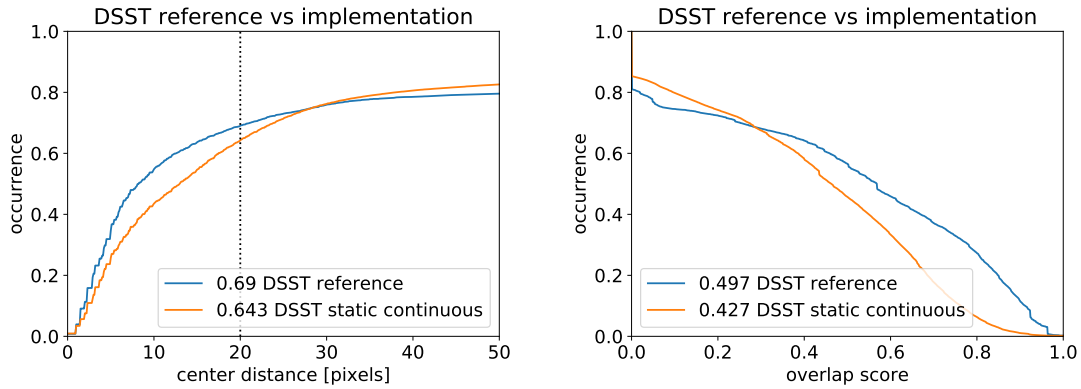
The most impactful parameters for this algorithm are the *inner_punish_threhsold* and *outer_punish_threshold* parameters, as they control under which conditions candidates are punished, thus increase or decrease the size of the final, scaled candidate. Inspecting exemplary results of the algorithm with the above parameter configuration revealed, that with an inner punishment threshold of 0.4 and an outer punishment threshold of 0.45, the algorithm achieved acceptable results on sequences where the object was gradually decreasing in size. The opposite case, where the object was gradually increasing in size, showed consistently bad results. The scale estimation algorithm would either not react to an increase in the size of the object and maintain a bounding box close to the initial size, or it would even produce a smaller bounding box. Exemplary size plots of such sequences are provided in Figure 5.2, which can be considered

representative for similar sequences. For the vast majority of sequences, an initial drop in size score could be observed, which further indicates that this parameter configuration might not be as ideal as it could be expected from the optimization.

The behavior in Figure 5.2 is likely to be caused by the values assigned to the inner and outer punish threshold. The values for the parameters have been optimized independently from one another, this means that the optimized parameter configuration does not necessarily contain the ideal parameter *combination*, instead it contains the *collection* of individual parameter values that achieved the highest score, when the other parameters had the values from the optimization baseline. Meaning that the best performing value for the inner punish threshold parameter was obtained when the outer punish threshold had a value of 0.5, just from the parameter optimization alone, nothing could be said regarding the performance of the actual combination of the parameter values.

5.2 Validation of the DSST algorithm

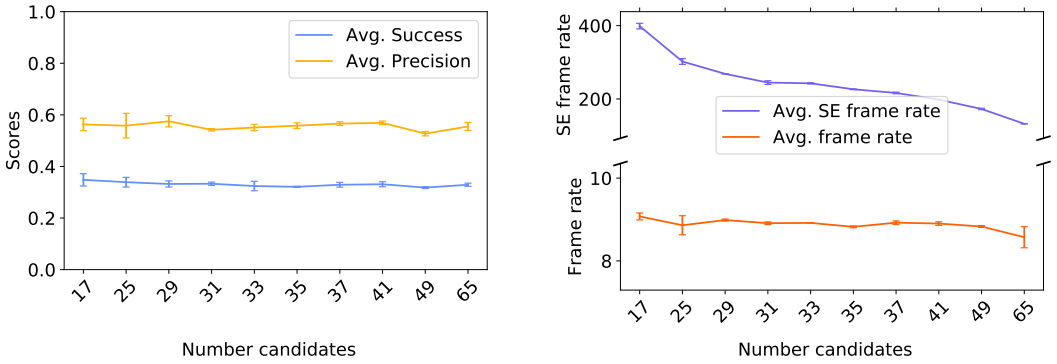
To ensure that the DSST algorithm by Danelljan et al. [9] has been implemented properly, the baseline version of the algorithm (i.e. without modifications regarding the aspect ratio) as implemented in the HIOB framework has been executed on the TB100 test set. As the quality of the output of the DSST algorithm is strongly sensitive to the positional prediction input, which is not deterministic in the HIOB framework, the experiment has been executed 10 times and the average values are reported. Like this, more robust and characteristic results can be obtained, which make for a fairer comparison between the reference implementation and the HIOB implementation. The comparison of the DSST reference implementation and the DSST implementation in HIOB is visualized in figure Figure 5.4, while the detailed sequence-wise results, are provided in the tables Table 5.1 and Table 5.2.



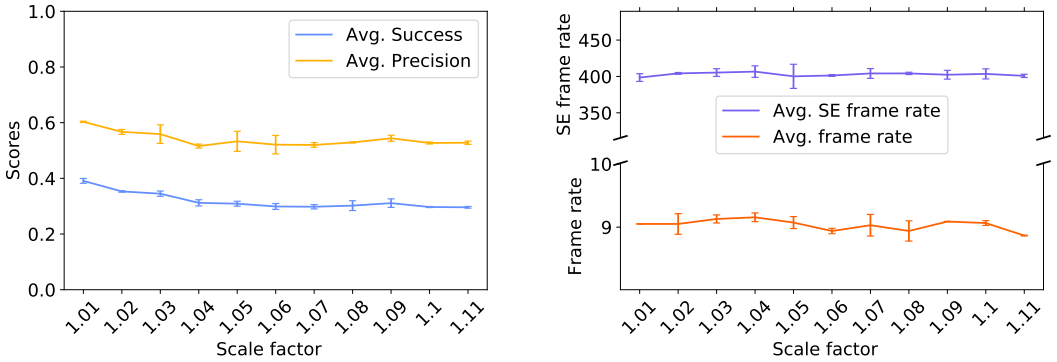
(a) The precision plot.

(b) The success plot.

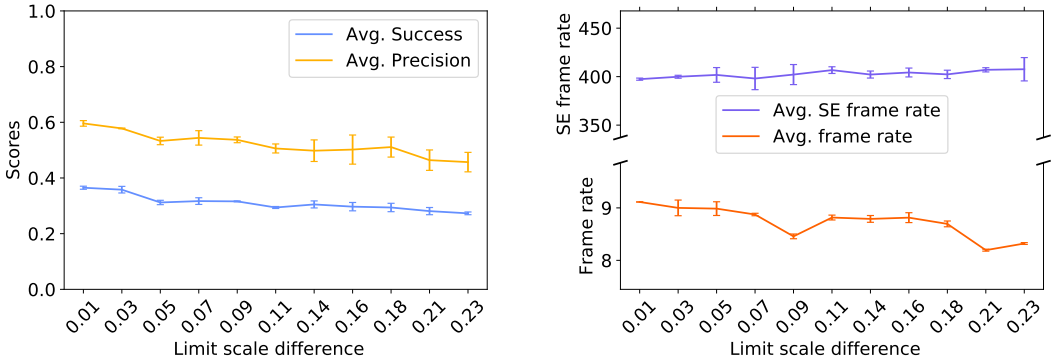
Figure 5.4: The plotted results of the DSST reference implementation from Danelljan et al. versus the static continuous DSST implementation in the HIOB framework.



(a) The average metric scores and the average impact on the computational load of the values of the *number_candidates* parameter. The best performing value is 17.0, with an average success rating of 0.348 and an overall tracking frame-rate of 9.075.

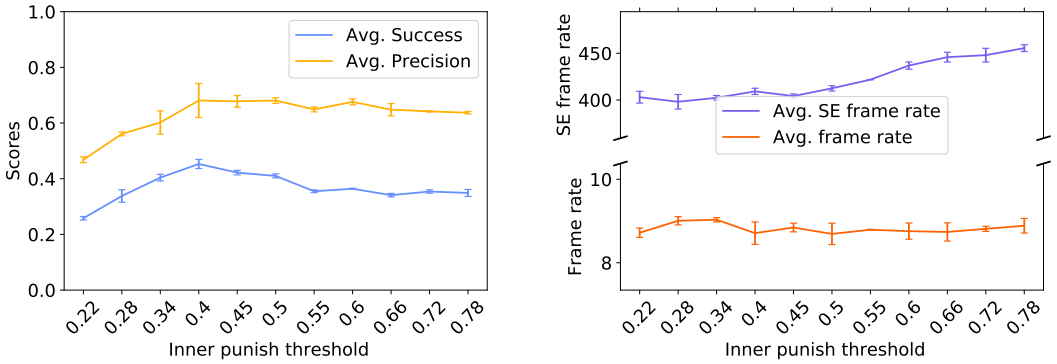


(b) The average metric scores and the average impact on the computational load of the values of the *scale_factor* parameter. The best performing value is 1.01, with an average success rating of 0.391 and an overall tracking frame rate of 9.051.

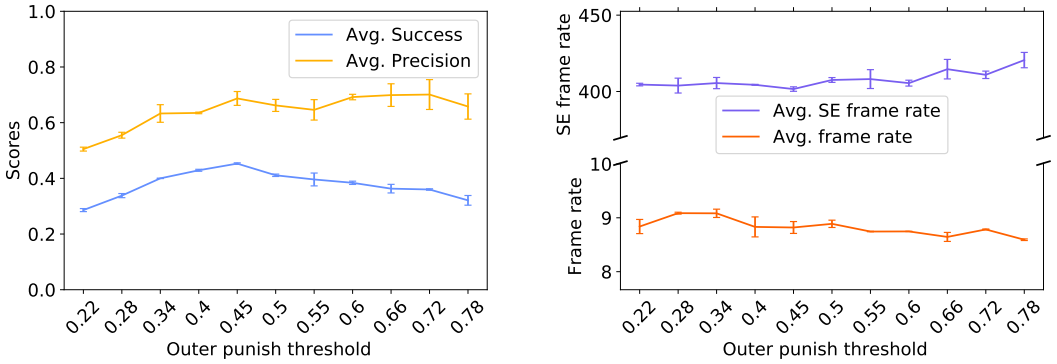


(c) The average metric scores and the average impact on the computational load of the values of the *limit_scale_difference* parameter. The best performing value is 0.01, with an average success rating of 0.365 and an overall tracking frame rate of 9.113.

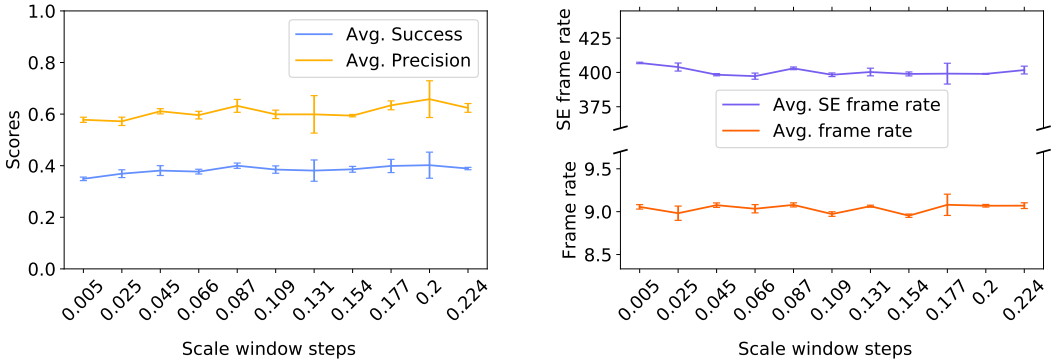
Figure 5.3: Results from the Scaled candidates parameter optimization.



(d) The average metric scores and the average impact on the computational load of the values of the *inner_punish_threshold* parameter. The best performing value is 0.4, with an average success rating of 0.453 and an overall tracking frame rate of 8.709.



(e) The average metric scores and the average impact on the computational load of the values of the *outer_punish_threshold* parameter. The best performing value is 0.45, with an average success rating of 0.453 and an overall tracking frame rate of 8.82.



(f) The average metric scores and the average impact on the computational load of the values of the *scale_window_step* parameter. The best performing value is 0.2, with an average success rating of 0.402 and an overall tracking frame rate of 9.069.

Figure 5.3: Results from the Scaled candidates parameter optimization (continued).

Sample	Frames	Precision	Success	Size Score
Basketball	725	0.903	0.552	20.32
Biker	142	0.444	0.266	27.13
Bird1	408	0.245	0.116	23.01
Bird2	99	0.545	0.6	8.88
BlurFace	493	1.0	0.895	14.29
Box	1161	0.369	0.337	25.76
Car1	1020	1.0	0.601	7.28
ClifBar	472	0.936	0.61	8.74
Crowds	347	1.0	0.748	7.11
David	471	1.0	0.797	5.98
Deer	71	0.789	0.602	17.92
Diving	215	0.228	0.198	23.65
DragonBaby	113	0.062	0.057	25.63
Dudek	1145	0.738	0.773	8.29
Freeman3	460	0.746	0.333	25.97
Liquor	1741	0.406	0.396	10.67
Panda	1000	0.214	0.122	11.7
RedTeam	1918	0.994	0.539	19.04
Singer2	366	0.915	0.751	10.1
Soccer	392	0.699	0.472	10.86

Table 5.1: The results of the DSST tracker by Danelljan et al. on the TB100 training subset.

Fig. 5.4 shows a considerable difference in both the achieved success rating and precision rating between the DSST reference version and the version as implemented in HIOB. To understand why the different implementations achieve different results, Table 5.2 and Table 5.1 need to be considered, which show the detailed results of the different algorithms for each sequence in the TB100 training subset.

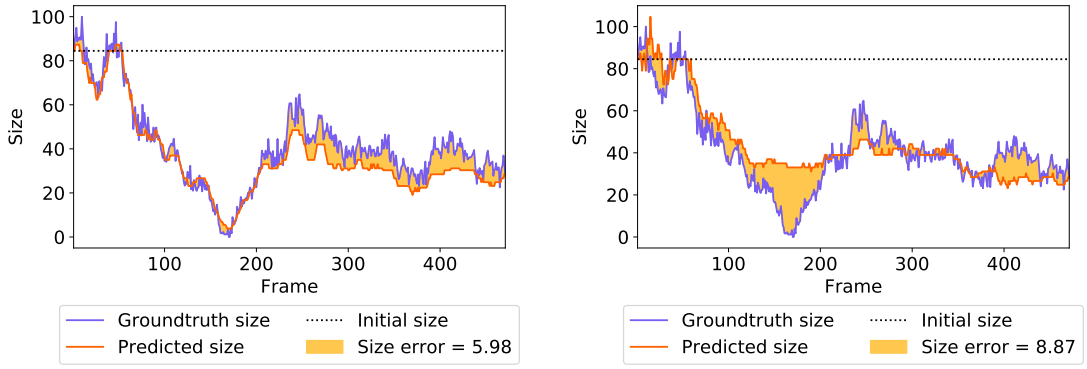
For the most part of this thesis, the precision metric is not of primary interest, because it measures the center distance between two bounding boxes, which is not directly affected by scale estimation. However, when a sequence has a low precision rating, the likelihood is high, that the tracker at some point lost the object. Even though precision is only of secondary interest, there is still an obvious interaction between precision and success, as the overlap between two bounding boxes is affected by the distance between them. This means, that for a fair comparison of the two DSST implementations, sequences should be considered that have a sim-

Sample	Frames	Precision	Success	Size Score
Basketball	725	0.354	0.196	438.837
Biker	142	0.88	0.499	30.211
Bird1	408	0.675	0.353	18.015
Bird2	99	0.974	0.678	35.688
BlurFace	493	0.371	0.496	44.959
Box	1161	0.558	0.481	32.703
Car1	1020	0.556	0.189	26.269
ClifBar	472	0.518	0.324	11.221
Crowds	347	0.999	0.643	28.645
David	471	0.991	0.669	10.49
Deer	71	0.777	0.574	67.827
Diving	215	0.244	0.123	47.058
DragonBaby	113	0.807	0.573	14.372
Dudek	1145	0.53	0.685	14.921
Freeman3	460	1.0	0.349	25.834
Liquor	1741	0.397	0.356	46.989
Panda	1000	1.0	0.593	12.643
RedTeam	1918	0.992	0.487	21.835
Singer2	366	0.045	0.065	55.651
Soccer	392	0.299	0.225	27.147

Table 5.2: The results of the HIOB tracker using the DSST static continuous scale estimation approach on the TB100 training subset.

ilar precision rating in booth trackers. As an extreme example, the DSST tracker achieves a very high precision rating on the sequence *Singer2*, while also having a high success rating and a low size score on the same sequence. Contrary, the HIOB tracker achieves a drastically low precision rating in the Singer2 sequence, which is in correspondence with a low rating in success and a high size score. Thus, using such a sequence to validate or invalidate the correct implementation of the algorithm would not be fair.

An opposing example is the sequence *David*, where both the DSST and the HIOB tracker achieve high precision and success ratings, while the achieved size score is for both trackers on the lower end, indicating good performance of the scale estimation algorithm. Figure 5.5 shows the plotted results if the size score of both trackers. Two things can be deducted from the comparison of the two plots. Firstly, both implementations achieve good results in estimating the scale of the object throughout the tracking sequence. Secondly, the DSST tracker a



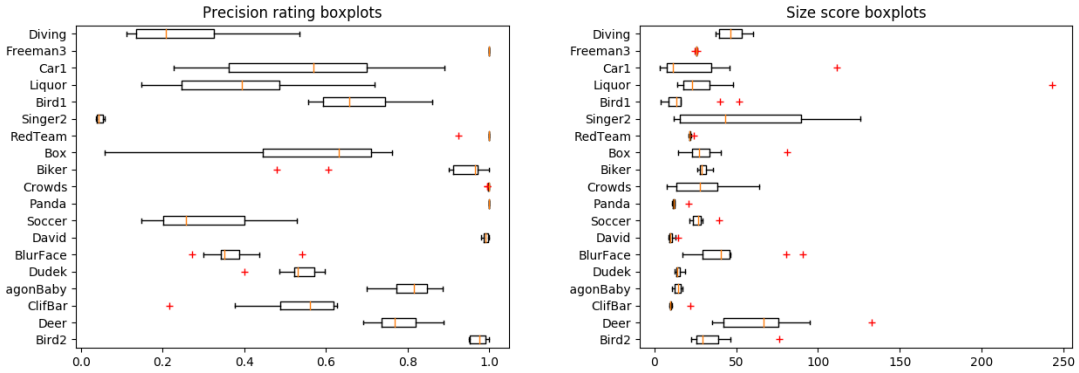
(a) The plotted size score of the DSST tracker by Danelljan et al.

(b) The plotted size score of the HIOB tracker using the implemented scale estimation DSST algorithm.

Figure 5.5: Comparrison of the size score between the DSST reference implementation by Danelljan et al. and the implementation in HIOB on the sequence David from the TB100 training subset.

achieves a slightly more accurate approximation of the actual ground truth scale than the HIOB implementation of the DSST scale estimation algorithm. It can not be said with absolute certainty what causes that behavior, but it is likely that the problem lies within the HOG descriptor because different implementations have been used. In the DSST tracker, the fast HOG implementation by Piotr Dollár is used (which is not available in Python) which implements HOG based as on the work of Felzenszwalb et al [10, 13]. In the HIOB implementation, the standard HOG descriptor provided by the OpenCV2 library is used, which implements HOG as described by Dalal et al. [10, 6]. As the HOG descriptor is run on every frame, the different HOG descriptors could explain why the DSST tracker achieves overall more accurate scale estimation results compared to the HIOB implementation of the DSST scale estimation algorithm.

While it has been shown that the HIOB implementation of the DSST scale estimation algorithm can achieve similarly good results (given comparable precision ratings) as the DSST tracker, cases should also be considered where the precision rating of the sequence is still acceptably high, but the success rating is low, indicating bad results from the scale estimation algorithm. This is the case for the sequence *Car1*, where the DSST tracker achieves strong results on the precision, success and size score metric, while the HIOB tracker with the DSST scale estimation implementation achieves overall worse results, with a drastically low success rating. To understand cases like this, a slightly more in depth statistical analysis of the 10 validation executions of the HIOB tracker has been conducted, by detecting outlier values in the precision and size score ratings, as visualized in Figure 5.6. The boxplot of the size score of the Car1 sequence shows generally low size scores and one outlier, which was not visible from only studying the average value over



(a) The precision ratings of the sequences over the 10 validation executions of the HIOB tracker visualized with boxplots.

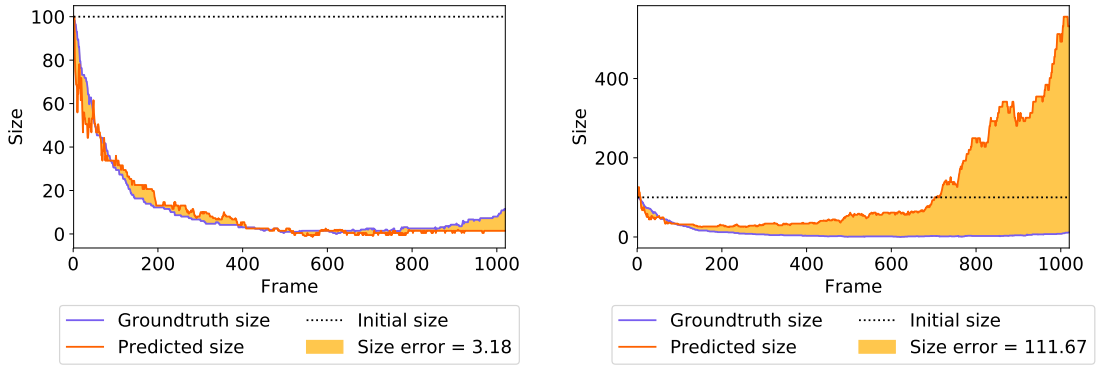
(b) The size errors of the sequences over the 10 validation executions of the HIOB tracker visualized with boxplots.

Figure 5.6: Boxplot visualization of the size error and the precision ratings of the 10 validation executions of the HIOB tracker for each sequence. The Boxplots show the median, first and third quantiles and outliers. The sequence Basketball has been excluded from the plots for better overview.

the 10 executions. This can be observed in Figure 5.7, which shows the outlier size plot compared to an average successfully size plot for the sequence Car1. This example also shows how sensible the algorithm is towards the predicted position, which is the main input to the algorithm on each frame. It is noteworthy that no outlier occurred in the precision ratings for the Car1 sequence, however, the boxplot shows a wide distribution of precision values. This means that the outlier size plot cannot be explained by bad translational predictions alone. Similar outliers can be observed in the boxplots of the sequences *Liquor*, *Box*, *Blurface* and *Deer*, which skewing the reported averages of the size scores in Table 5.2.

The sequence *Freeman3* shows similar characteristics as the Car1 sequence, that is having a high precision but low success rating, and a relatively high size score. However, this is the case in both the DSST tracker and the HIOB tracker, which allows the conclusion that the Freeman3 sequence has some characteristic that is challenging for the DSST scale estimation algorithm, independently of the tracking framework. In fact, the initially low resolution of the target cannot be well described with HOG, which explains the poor performance.

To conclude this section, the DSST tracker and the HIOB tracker with the DSST scale estimation implementation achieve similarly good results when the precision ratings are also similar, which can be seen seen on sequences like Biker, Cliffbar, David, Dudek or Readteam, while Danelljan et al.s DSST tracker achieves generally slightly better size score, which is likely to be caused by the different HOG descriptor implementations.



(a) The size error of a good execution.

(b) The size error of the outlier execution.

Figure 5.7: Sizes cores from different execution of the 10 validation executions, showing how the different predicted positions can affect the outcome of the algorithm.

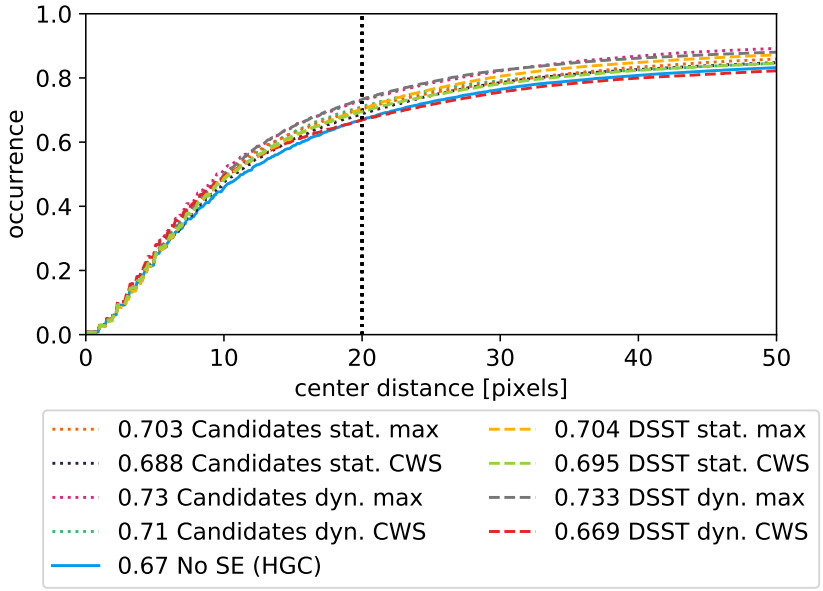
5.3 Performance on the TB100 dataset

This experiment aims at exploring the results the different algorithms achieve on the diverse TB100 dataset.

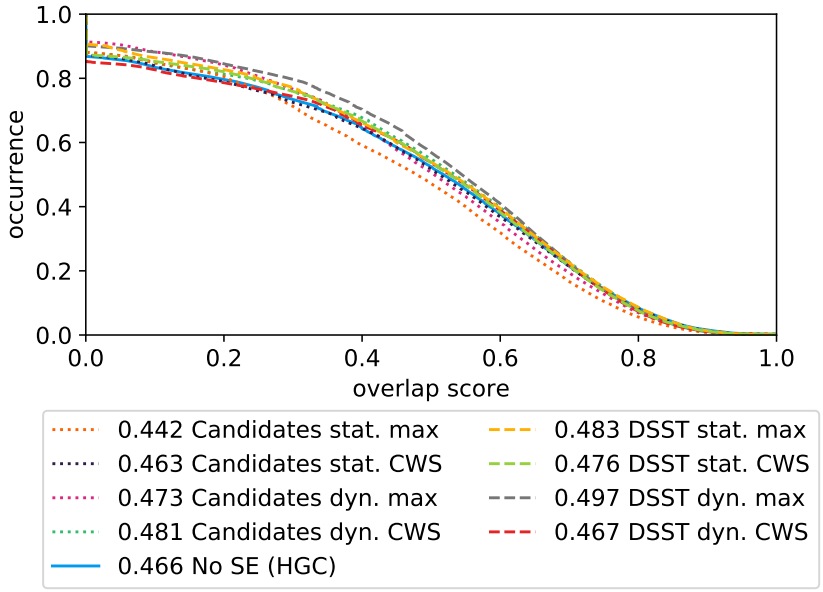
In total, there are eight different algorithm versions and a baseline, where the SE module has been disabled. The eight different algorithm versions are the product of the main two algorithms (Candidates and DSST) which both have a static and dynamic aspect ratio implementation, and the two, SE specific update strategies *Max* and *CWS*. To avoid confusion regarding the update strategies: The HIOB tracker is configured to always use the HGC update strategy, as the update strategy achieved the best results in the work of Springübe and in the work of Heinrich et al. [30, 17]. As explained in section 4.4, the update strategies used in this thesis only control when the SE module is executed, instead of controlling the model update behavior of the entire tracker. This enables a continuous calculation and adjustment of the scale, and once the HIOB updater HIOBs internal representation of the model, the scaled representation is learned.

The different algorithm versions have been executed on the complete TB100 dataset. As the TB100 features a very diverse set of sequences, the results of the algorithms on the TB100 represent rather general performance, instead of domain-specific performance, as explored on the NICO dataset.

The results of the algorithms on the TB100 dataset are visualized in Figure 5.8, showing results of the precision and success plots of each algorithm. The results of the versions of the DSST and Scaled candidates algorithms that achieved the highest success scores are in greater details shown in Table 5.3. The HIOB tracker without the use of the SE module is provided as the baseline.



(a) The precision plots of the two algorithms and their variations on the TB100 dataset with adjusted center distance threshold.



(b) The success plot of the two algorithms and their variations on the TB100 dataset. The DSST algorithm achieves best results with the Full update strategy and the dynamic aspect ratio implementation. The Candidates algorithm achieves best results with the HGC update strategy and the dynamic aspect ration implementation.

Figure 5.8: The plotted results of the different algorithms on the TB100 dataset. Both algorithms use the parameter configuration that allows the scale to be changed.

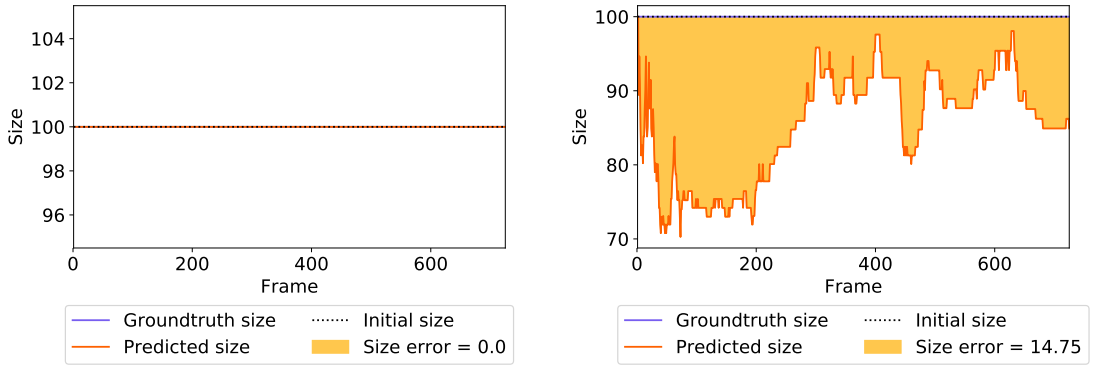
Algorithm	No SE (HGC)			Candidates dyn. CWS			DSST dyn. max		
Attribute	Precision	Success	Size	Precision	Success	Size	Precision	Success	Size
IV	0.668	0.466	30.713	0.695	0.484	30.934	0.711	0.485	31.909
SV	0.622	0.415	31.254	0.642	0.423	31.089	0.672	0.443	29.267
OCC	0.598	0.428	23.891	0.639	0.454	24.503	0.637	0.443	33.801
DEF	0.620	0.428	25.696	0.663	0.45	26.736	0.650	0.442	33.855
MB	0.541	0.446	25.020	0.569	0.472	25.224	0.561	0.438	30.574
FM	0.599	0.467	22.183	0.613	0.476	22.637	0.599	0.448	29.047
IPR	0.688	0.482	23.567	0.708	0.494	23.937	0.724	0.493	29.981
OPR	0.678	0.467	24.883	0.700	0.481	25.488	0.723	0.483	28.558
OV	0.538	0.404	17.507	0.608	0.438	18.871	0.585	0.428	29.259
BC	0.613	0.437	26.332	0.696	0.475	26.587	0.662	0.446	29.585
LR	0.815	0.389	32.836	0.774	0.377	32.308	0.869	0.433	22.337
All	0.67	0.466	26.672	0.71	0.481	27.286	0.733	0.497	32.032

Table 5.3: The results on the TB100 of the best performing algorithm versions in comparison to the HIOB baseline without the SE module.

Considering Figure 5.8, it is apparent, that the results of the two algorithms are only slightly better than the baseline HIOB tracker, which employs no SE module and uses the HGC update strategy. The HIOB baseline achieves a total success rating of 0.466 on the complete TB100 dataset, while the best performing version of the Candidates algorithm achieves a total success rating of 0.478 and the best performing Version of the DSST algorithm achieves a success rating of 0.497.

It also needs to be addressed, that for this experiment, each algorithm has only been executed on the dataset once (contrary to the DSST validation experiment, where the results were reported based on the average values of 10 separate). This means, that the achieved results are likely to be influenced by the slightly different translational predictions of the HIOB tracker, which can have a drastic impact on the performance of the DSST algorithm, as it has been shown in Figure 5.7. The Candidates algorithm is expected to be less sensitive towards slightly different translational predictions because it depends on HIOBs feature map, which is at a low resolution due to the pooling layers in the CNN.

Table 5.3 reveals interesting relations between the performance of the different algorithms on the different attributes of the TB100 dataset. Most notably is, that the No SE baseline condition actually has the lowest average size score compared to best performing versions of Candidates and DSST algorithms. This means, that the average predicted scale of the HIOB baseline was closer to the ground truth scale of the object than the predicted scales of the SE module was close to the ground truth scale of the object. This can be explained by the following observations. Firstly, in the TB100 exist a lot of sequences in which the ground truth bounding box never changes its size. On such sequences, the No SE HIOB baseline achieves the best size score possible, because the size is equal to the initial size of the object throughout the entire sequence. On such sequences, the slightest errors of the SE



(a) An unrealistic size plot where the ground truth bounding box never changes it size and thus, the No SE condition achieveing a perfect size score rating.

(b) The size plot of the DSST algorithm with dynamic aspect ration implementation and the Max update strategy, which achieves a worse size score.

Figure 5.9: The size plots of the TB100 Basketball sequence, where the ground truth bounding box never changes its size.

module immediately result in worse performance compared to not updating the size. An example of this scenario where the HIOB baseline without the SE module has an unfair advantage against the SE algorithms is visualized in Figure 5.9.

Additionally, in their current state, both SE algorithms are not flawless, meaning that there are sequences where they will fail. This failing under certain conditions (like no ideal predicted position), in combination with the ideal size scores achieved by the HIOB baseline on other specific sequences, are likely to outweigh the cases in which the SE module achieves strong results at estimating the scale.

Further, the TB100 attribute of special interest is the Scale Variation (SV) attribute. Table 5.3 shows very similar success ratings for the No SE baseline and the Candidates algorithm, while the DSST algorithm achieves a higher success rating when only the sequences that have the SV attribute are taken into consideration. This almost identical success rating of the baseline and the Candidates approach can be explained by the update strategy of the Candidates algorithm and the parameter configuration from the optimization. The update strategy of the best performing Candidates version is CWS, meaning that the scale is only updated on specific frames, but at least once every 20 frames. The update strategy in combination with the relatively high value of 0.2 for the *Scale window step size* results in a setting, in which the algorithm depends on very strong indications on the prediction map, as otherwise, the punishment of the divergence to the unchanged size will outweigh a candidate that achieves only slight improvements. Thus, the algorithm punishes scale change a lot on the frames in which the algorithm is actually executed, resulting in a greatly decreased likelihood for the algorithm to change the scale. This means that the high values of the Candidates algorithm with the CWS update strategy are likely to be the product of not changing the scale to the

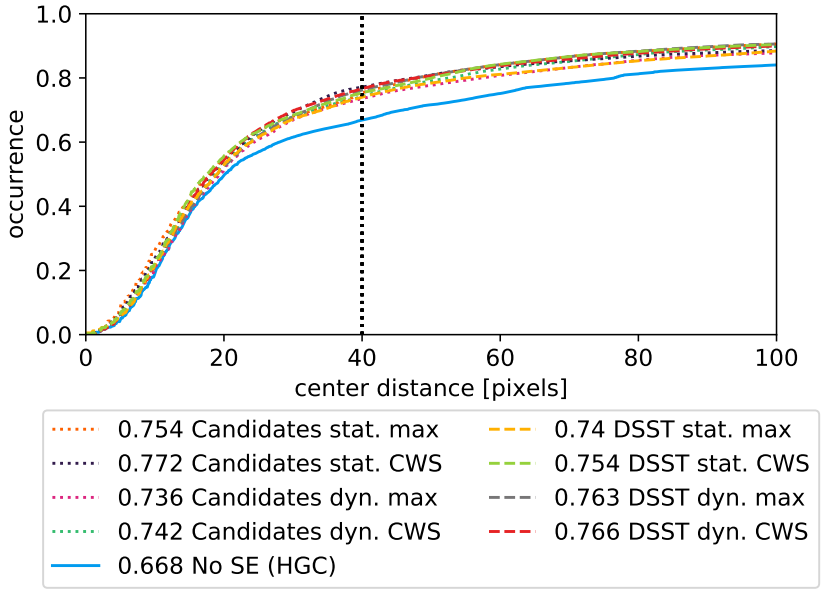
worse on bad frames instead of changing the scale for the better on good frames.

The DSST algorithm achieves better results than the HIOB baseline on the SV sequence collection of the TB100, which is plausible because the algorithm's goal is to estimate the size during tracking. It needs to be noted that on the SV sequence collection, the DSST algorithm condition also achieved a higher precision rating that both the HIOB baseline and the Candidates algorithm. This means that to some degree, better success ratings must be attributed to better translational predictions. It should also be highlighted, that for most attributes, the DSST algorithm has a higher size score than the HIOB baseline, which is likely to be the result of bad result on a few sequences, which skew the average values, and the previously described phenomenon that on some TB100 sequences, the ground truth bounding box is of static size.

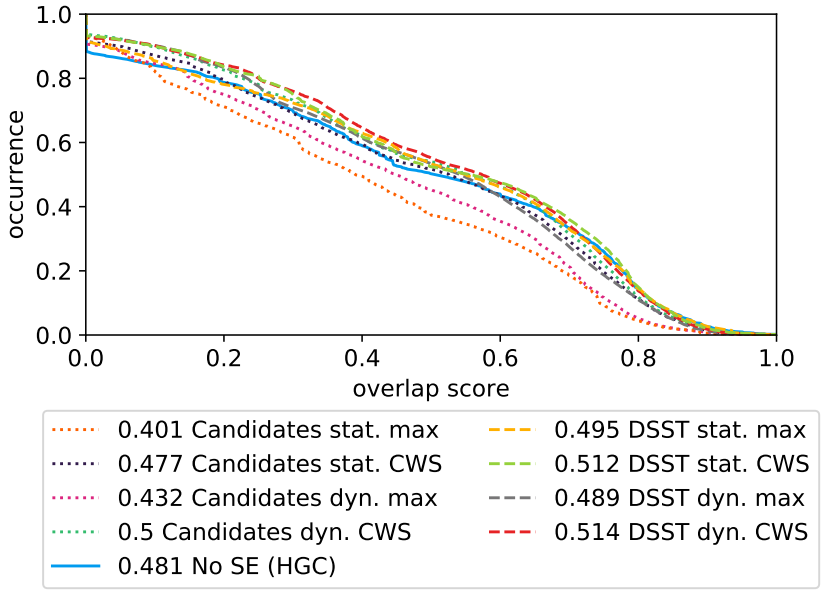
The interesting question arises, why the achieved success ratings of the DSST algorithm are not significantly higher for at least the TB100 sequences with the SV attribute. This is explained best with example sequences, and the question will be answered in section 5.6.

5.4 Performance on the NICO dataset

Analyzing the results of the algorithms on the TB100 dataset revealed the performance of the algorithms on diverse sequences. This experiment explores how the scale estimation algorithms perform on the specialized NICO dataset. This represents a real-world robot interaction task setting. Note, that the NICO dataset is completely recorded in high definition, contrary to the TB100, and thus the threshold value for the precision metric has been increased from 20 pixels to 40 pixels, in accordance with Heinrich et al.'s reported results on the NICO dataset [17].



(a) The precision plots of the two algorithms and their variations on the NICO dataset with adjusted center distance threshold.



(b) The success plot of the two algorithms and their variations on the NICO dataset. The DSST algorithm achieves best results with the HGC update strategy and the dynamic aspect ratio implementation. The Candidates algorithm achieves best results with HGC update strategy and the dynamic aspect ration implementation.

Figure 5.10: The plotted results of the two algorithms on the NICO dataset. Both algorithms use the parameter configuration that allows the scale to be changed.

Algorithm	No SE (HGC)			Candidates dyn. CWS			DSST dyn. CWS		
Attribute	Precision	Success	Size	Precision	Success	Size	Precision	Success	Size
bright	0.591	0.54	29.859	0.607	0.549	37.073	0.607	0.561	35.978
size-change	0.416	0.426	18.484	0.462	0.437	20.312	0.484	0.463	19.218
occlusion	0.391	0.431	21.688	0.475	0.498	25.706	0.478	0.49	25.326
dark	0.546	0.525	30.957	0.588	0.54	38.529	0.616	0.556	35.635
motion-blur	0.322	0.398	29.315	0.359	0.421	31.952	0.394	0.446	29.761
part-occlusion	0.351	0.413	30.463	0.397	0.429	37.223	0.406	0.448	33.033
non-square	0.338	0.286	29.675	0.366	0.299	31.468	0.353	0.324	29.828
contrast	0.589	0.642	25.841	0.541	0.639	32.676	0.596	0.642	35.134
All	0.496	0.481	30.262	0.535	0.499	37.607	0.545	0.514	35.852

Table 5.4: The results on the NICO dataset of the best performing algorithm versions in comparison to the HIOB baseline without the SE module.

Considering the results the different algorithms achieved on the NICO dataset, the Candidates algorithm with the Max update strategy obtained significantly worse results than the Candidates algorithm with the CWS update Strategy. This is most likely because when the NICO robot interacts with an object, it occludes the object with its hand, which causes the Candidates algorithm, when executed on every frame, to adjust to the occluded object by decreasing the size to the remaining part of the object. As the configuration of the Candidates algorithm struggles with increasing the object size, it is likely that the predicted size will not be readjusted once the object is no longer occluded by the robot's hand.

A final, in-depth analysis of the behavior of the different algorithms on exemplary sequences from both datasets is provided in section 5.6.

5.5 Realistic constraints and the computational load

This section analyses the computational load of the different algorithms, with the goal of determining whether an algorithm is suitable in a real-world setting. The computational load of the different algorithms is measured by their impact on the overall frame rate of the HIOB tracker, while the achieved frame rates of the isolated algorithms are also taken into consideration. Table 5.5 shows the frame rates of the different algorithms on the NICO dataset, which shows multiple trends in the frame rate data. The NICO dataset is more relevant for this analysis, because the NICO sequences show footage of the real world use case with the NICO robot, compared to the diverse sequences of the TB100.

The baseline HIOB tracker without the SE module achieves an overall frame rate of 5.841 FPS and, while the lowest overall frame rate is achieved by the DSST algorithm with the dynamic aspect ration implementation and the Max update strategy. It should be noted that the overall frame rate of the HIOB tracker was

slightly higher on the TB100 dataset with 7.86 FPS for the No SE condition. The difference in the FPS between the two datasets is caused by the fact that the sequences of the NICO dataset have all been recorded at a resolution of 960×720, while most TB100 sequences are of lower resolutions like 640×480 or 320×240.

A much wider FPS range can be observed in the isolated FPS ratings of the SE module, which measures only the frame rate of the specific SE algorithm and excludes the other HIOB modules that contribute to the tracking process from the calculation. The FPS data shows that the Candidates algorithm achieves drastically higher frame rates than the DSST algorithm, while for both algorithms, the dynamic versions (dynamic aspect ration) of the algorithms achieve significantly lower frame rates than the static versions (static aspect ration) of the algorithms. This is not surprising, as the dynamic versions of both algorithms have been implemented by simply applying the algorithm once on the x-axis and once on the y-axis of the object. This directly increases the time complexity from $\mathcal{O}(n)$ to $\mathcal{O}(2n)$ for both algorithms, where n is the number of images in a sequence.

Finally, it can be observed that the CWS update strategy achieves drastically higher frame rates than the Max update strategy for the SE module, independently of the algorithm. This is also not surprising because the CWS update strategy skips the execution of the SE algorithm when HIOBs confidence rating on a frame is lying out of the confidence window.

Recalling that the computational load of the algorithms is a deciding factor between whether an algorithm is suited for application in a real-world setting or not, a relative comparison is not enough. Instead, the absolute achieved frame rates need to be considered. Heinrich et al. report 11-17 FPS of the HIOB tracker in combination with the NICO robot, which will be used as a baseline FPS value for comparison [17]. The absolute difference in the FPS values reported by Heinrich et al. and the frame rates achieved in this thesis can be attributed to different versions and configurations of the HIOB tracker.

The absolute achieved frame rates of the SE module indicate, that only the DSST algorithm introduces significant additional computational load. Further, the CWS update strategy skips the execution of the SE algorithm on enough frames, which compensates for the costly execution of the algorithm. Thus, the DSST algorithm in combination with the CWS update strategy poses no significant additional computational load either. However, executing the DSST algorithm on every frame (aka using the Max update strategy) causes a significant reduction of the frame rate. To obtain a better measurement of the computational load associated with an algorithm, the overall frame rate of the HIOB tracker with the different algorithms is normalized between 0 and the FPS achieved by the No SE condition (5.841 FPS). This produces a factor for each algorithm, that indicates the speed of an algorithm compared to the No SE baseline condition. For example, the most costly algorithm (DSST with dynamic aspect ratio and Max update strategy) run at 58.4% the speed of the baseline HIOB tracker without scale estimation.

The high cost associated with the DSST algorithm in combination with the Max update strategy renders the algorithm highly unfavorable for real-world applications. Even though the DSST algorithm achieves slightly better results with

Algorithm	Overall FPS	Norm. overall FPS	SE FPS	Success	Size error
Candidates stat. Full	5.529	0.947	487.093	0.401	73.705
Candidates stat. HGC	5.69	0.974	3237.84	0.508	35.905
Candidates dyn. Full	5.703	0.976	330.096	0.432	61.496
Candidates dyn. HGC	5.753	0.985	2574.431	0.507	33.712
DSST stat. Full	4.71	0.806	26.557	0.495	43.122
DSST stat. HGC	5.906	1.011	388.972	0.512	38.828
DSST dyn. Full	3.409	0.584	8.391	0.488	40.996
DSST dyn. HGC	5.724	0.98	140.382	0.514	35.852
No SE	5.841	1.0	-	0.481	30.262

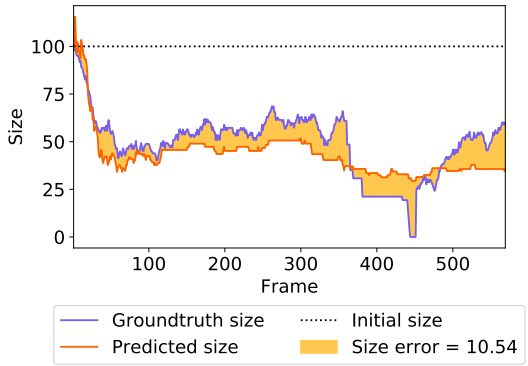
Table 5.5: Framerates of overall tracker and isolated SE module on both datasets

the CWS update strategy than with the Max update strategy on the NICO dataset, the generalization that the DSST algorithm always performs better with the CWS update strategy cannot be made, because on the TB100 dataset, the opposite is true and the DSST algorithm with the Max updates strategy achieves better results than the DSST algorithm with the CWS update strategy. Thus, the DSST algorithm with the dynamic aspect ration implementation and the Max update strategy must be categorized as too costly for real-world implications. The DSST algorithm with the static aspect ratio implementation and the Max update strategy is still rather costly with a frame rate of 80.0% compared to the HIOB baseline without a SE module, is still rather costly, but could be justified with a significant increase in the results.

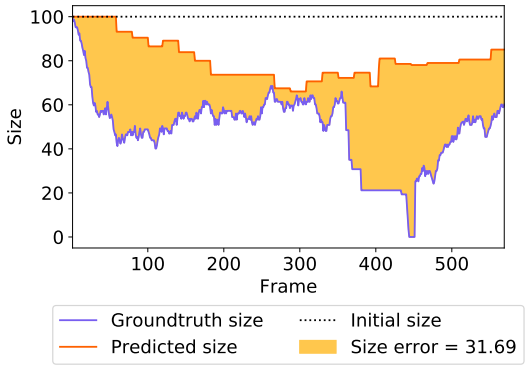
Summarizing the section, it is clear that independent of the static or dynamic aspect ratio implementation and independent of the update strategy, the Candidates algorithm introduces very little additional computational load and is suited for real-world application, while the DSST algorithm is rather computationally demanding and depends on the CWS update strategy to reduce its computational load so that the algorithm can be used for real-world applications.

5.6 In depth sequence analysis

This section finally provides an in-depth analysis of the behavior of the implemented algorithms, by investigating exemplary sequences. It is of main interest to understand under which conditions the different algorithms achieve good results, and when under which they fail. Thus, representative good and bad sequences are equally analyzed for the Candidates and DSST algorithm. Firstly, general performance on the full TB100 dataset is analyzed and secondly, the capabilities of the SE algorithms to deal with the typical challenges introduced on the NICO dataset are investigated.



(a) The size plot of the DSST algorithm with dynamic aspect ratio implementation using the Max update strategy, showing an accurate scale estimation.



(b) The size plot of the DSST algorithm with dynamic aspect ratio implementation using the CWS update strategy. The estimated scale is less accurate, because the execution of the algorithm has been skipped due to strong results from the visual features at unchanged size.

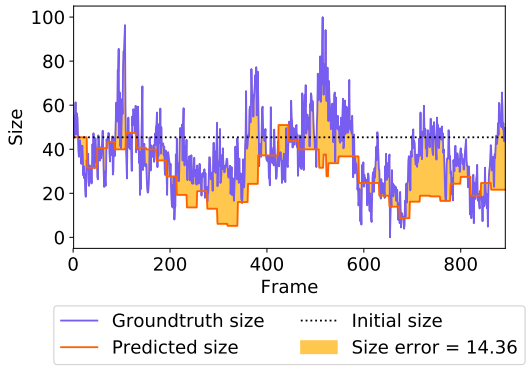
Figure 5.11: The size plots of the DSST algorithm on the TB100 *Trellis* sequence showing how the CWS update strategy can result in less accurate scale estimation results.

5.6.1 Performance on TB100 sequences

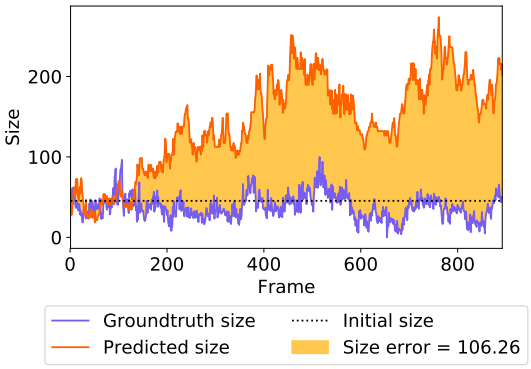
The DSST algorithm has already shown to be capable of correctly estimating the scale of an object in the context of the validation experiment. Similarly, good results are present in the final results on the complete TB100. It is interesting to see how the CWS update strategy can affect the outcome of the algorithm because the CWS update strategy should prevent the algorithm from updating the scale on bad frames and thus achieve more robust results. A comparison between the size plots of the DSST algorithm using the Max and the CWS update strategy is provided in Figure 5.12. The size plot of the DSST algorithm with the dynamic aspect ratio reveals an initial jump at the 120-*th* frame, with a second, more severe jump in size at the 400-*th* frame mark. To gain a better understanding of the events that cause the corruption of the scale mode, the key frames have been extracted and are analyzed in Figure 5.12.

Analyzing the key frames of the sequence *FaceOcc1* revealed, that the CWS update strategy benefits the DSST algorithm because it skips execution of the algorithm on bad frames, which would result in an inaccurate scale prediction. Preventing such updates is crucial because if a bad scale is predicted even once, it can potentially cause HIOBs internal representation model to get corrupted.

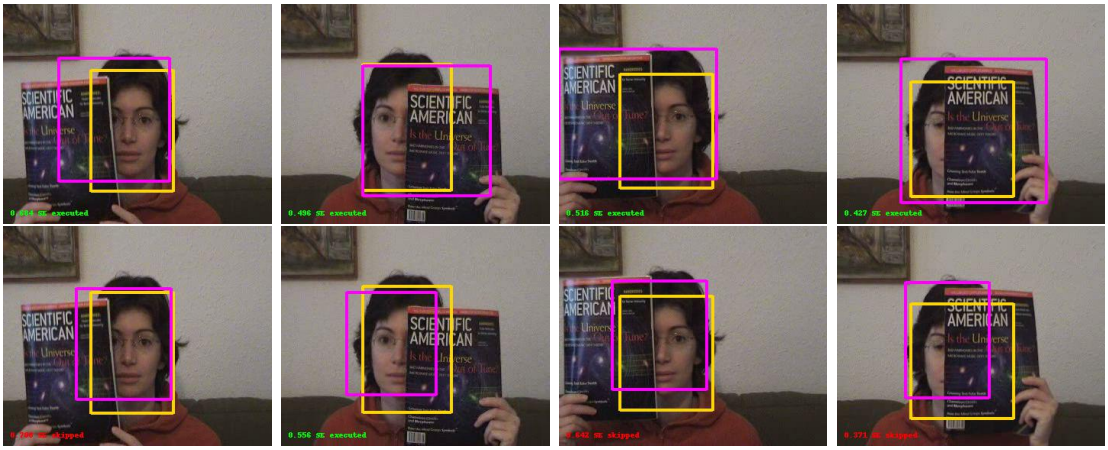
Considering how well the CWS update strategy works with the DSST algorithm on the TB100 sequence *FaceOcc1*, the question arises why the CWS update strategy does not consistently achieve significantly better results than the Max



(a) The size plot of the DSST algorithm with dynamic aspect ratio implementation using the Max update strategy.



(b) The size plot of the DSST algorithm with dynamic aspect ratio implementation using the CWS update strategy.



(c) Frame 139, with the Max update strategy (top) the model is partly trained on the journal, while with the CWS update strategy (bottom) execution of the SE module is skipped.

(d) Frame 225, with the Max update strategy (top), the journal is completely enclosed in the bounding box, while with the CWS update strategy (bottom), the journal is still excluded from the prediction.

(e) Frame 385, with the Max update strategy (top), journal and face appear to be equally considered as the target. The CWS update strategy(bottom) prevents updating the scale, because the visual features corresponding to the face are still strong enough.

(f) Frame 523, the Max update strategy (top) trains on the complete journal. The CWS update strategy (bottom) prevents scale estimation under significant occlusion.

Figure 5.12: The size plots and key frames of the TB100 *FaceOcc1* sequence. Better results are achieved by the CWS update strategy (bottom row), because the SE module is not executed under partial occlusion. The ground truth bounding box is annotated in yellow and the predicted bounding box in magenta.

update strategy with the DSST algorithm. As indicated earlier, this can partly be related to different translational predictions, as the results presented in Figure 5.8 are all based on one tracking execution. However, what is also likely contribute to the overall worse result of the DSST algorithm with the CWS update strategy on the TB100 is, that slight changes in the scale are ignored because the overall features are still good enough and updating HIOBs internal object representation at the slightly different scale is not necessary. How the CWS update strategy can cause the SE algorithm to achieve worse results is shown in Figure 5.11.

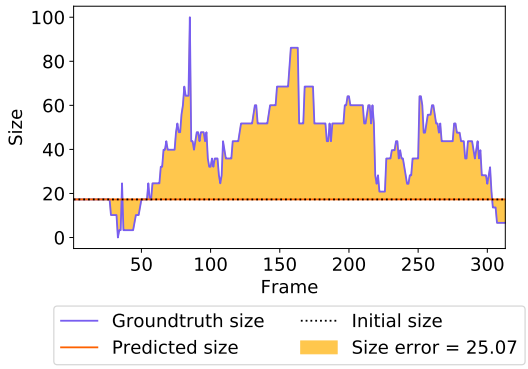
The examples given above indicate, that the DSST algorithm is certainly able to produce accurate scale estimation results. Using the algorithm does however not appear to greatly increase the overall tracking results, as shown in Figure 5.8. Thus, sequences have been identified with the similar precision rating but significantly different size scores and or success ratings, because under those conditions, it is likely that the results have been negatively affected by the SE module. To keep this section short, only the results achieved by the best performing version of the DSST algorithm (dynamic aspect ratio, Max update strategy) on the TB100 dataset are considered and compared to the results achieved by the HIOB baseline without the SE module.

One sequence that shows the negative consequences of employing the SE module with the DSST tracker (dynamic aspect ratio, Max update strategy) is the TB100 *Jumping* sequence. In the HIOB baseline version, a precision rating of 0.996, with a success rating of 0.561 and a size score of 25.07 has been achieved on the Jumping sequence. With the use of the DSST algorithm, a precision rating of 1.0, a success rating of 0.455 and a size score of 62.25 has been achieved, which is significantly worse than the baseline results. Comparing the size plots in Figure 5.13, it can be seen that the DSST algorithm significantly decreased the scale factor at the beginning of the sequence, and never corrected the scale factor throughout the rest of the sequence. Detailed inspection of the frames in the sequences revealed, that an interlacing technique[2] has been used on the sequence, which means that edges from two frames are extracted at all times. Under such conditions, it is not surprising that a feature extractor that operates based on edge detection is not performing well, and the bad results of the SE module can be attributed to this observation.

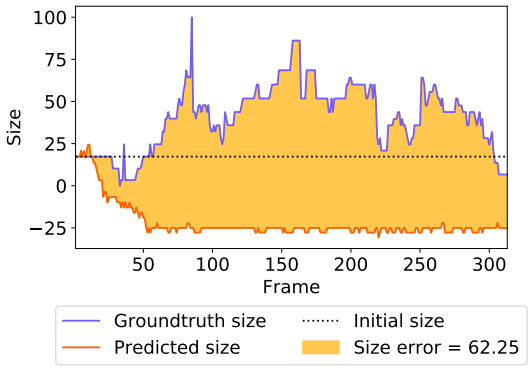
Generally, sequences of low resolution have been found to be problematic for the DSST algorithm. This confirms a weakness that has also been reported by Danelljan et al. [9]. Danelljan et al. reason, that this is due to the HOG feature-descriptor, which performs poorly at low resolutions. This seems plausible because the lower resolution of a sequence is, the less clear can the edges be expected to be.

The TB100 *Jumping* sequence describes the worst case, where the HIOB baseline without the use of the SE module achieves higher results than HIOB with the SE module. There are different examples that could be given, where better

[2]Interlacing doubles the perceived frame rate (temporal resolution) of a video by combining signals from two frames into one.

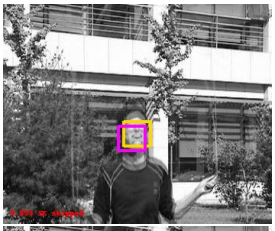


(a) The size plot of the No SE HIOB baseline

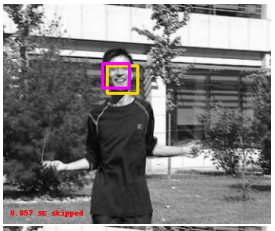


(b) The size plot of the DSST algorithm with dynamic aspect ratio implementation using the Max update strategy.

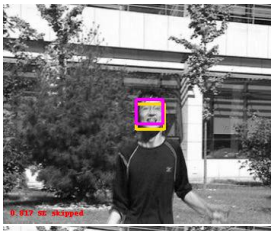
(c) Frame 10, initially the bounding boxes are aligned. The strong edges in the earlier frames are not problematic for the DSST algorithm.



(d) Frame 54, with the HIOB baseline (top), the overlap between the bounding boxes is still high, while with usage of the SE module, the bounding box has been decreased in size



(e) Frame 181, overlap of the bounding boxes is still high in the HIOB baseline (top), while with the SE module the model representation is now trained on the mouth and nose.



(f) Frame 224, overlap in the HIOB baseline remains high (top). Overlap with the SE module remains low, as the object representation has been decreased to a smaller part of the face.

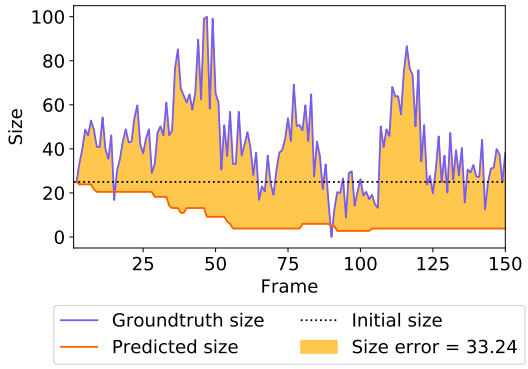
Figure 5.13: The size plots and key frames of the TB100 *Jumping* sequence. Better results are achieved by the HIOB baseline. The ground truth bounding box is annotated in yellow and the predicted bounding box in magenta.

results are obtained from not using the SE module. Generally, when there is only little scale variation in a sequence and the sequence has attributes that have been identified to be problematic with the DSST algorithm (like occlusion, interlacing techniques or very low resolution), the likelihood is high that not using the SE module will yield better results.

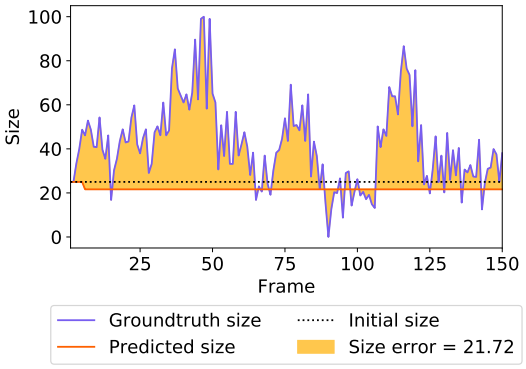
The best performing version of the Candidates algorithm on the TB100 datasets uses the dynamic aspect ratio implementation and the CWS update strategy. Investigating the results of said version of the algorithm revealed, and fails to correctly react to even significant scale changes under favorable condition (i.e. no occlusion and relatively high sequence resolution). The results show, that the algorithm only occasionally (partly because of the update strategy) adapts the scale slightly in the correct direction, but does not accurately update the scale throughout the sequence, an example for this is given in Figure 5.15. This explains why the algorithm still achieves slightly better results than the HIOB baseline, but the difference between success rating of the HIOB baseline and the best performing version of the Candidates algorithm (0.466 and 0.481) are insignificant. It must also be noted, that the slightly different translational predictions are still a contributing factor in those scores, which further indicates that the Candidates algorithm with the CWS update strategy is in its current state not achieving desirable results.

This must, to a certain degree, be attributed to the parameter configuration that has been obtained in the optimization experiment. The optimization experiment has been conducted using the Max update strategy with the static aspect ratio implementation, as this can be considered as the most basic version of the algorithm. For the Max update strategy, the obtained parameter configuration makes sense. For example, the value of the *Scale window step size* parameter has been set to 0.2, which means that each candidates punishment score gets increase by 0.2%, based on how many scale steps it is off the unchanged scale factor. Such a high value for this parameter makes sense when the algorithm is executed on every frame, as it prevents bad frames from immediately corrupting the representation model and generally corrects the output of the algorithm to bot change the scale on the smallest changes. For the CWS update strategy, however, a different, less extreme value is probably adequate, which would have to be confirmed or dis-proofed by a specific experiment.

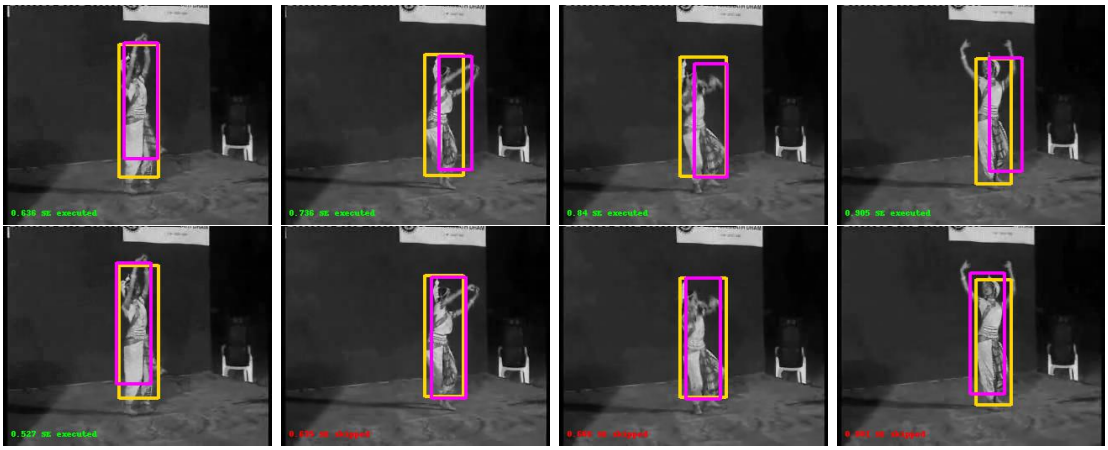
On the TB100 *Dancer2* sequence, the Candidates algorithm with the CWS update strategy and the Candidates algorithm with the Max update strategy achieved similar precision ratings (1.0 and 0.99), with divergence in the success rating (0.71 and 0.652) and size score (21.72 and 33.24), which indicates bad performance of the Candidates algorithm with the Max update strategy. Comparing the size plots of the TB100 *Dancer2* sequence for the Candidates algorithm using the Max and the CWS update strategy in Figure 5.14, it is observable, that with the Max update strategy, the size has been continually reduced throughout the first 50 frames, which is never corrected. This is to a certain degree in accordance with the finding from the optimization, where the problem of the Candidates algorithm with the Max update strategy has been identified, that describes the shrinking of the bounding box at the beginning of the sequence.



(a) The size plot of the Candidates algorithm with the Max update strategy and dynamic aspect ratio implementation.



(b) The size plot of the Candidates aspect ratio with the CWS update strategy and dynamic aspect ratio implementation.



(c) Frame 39, with the Max update strategy (top), the size has already been decreased, due to low values at the outer ends of the object on the prediction mask. With the CWS update strategy (bottom), this effect has been mostly suppressed.

(d) Frame 60, with the Max update strategy (top), the size of the bounding box has been further decreased, while with the CWS update strategy (bottom), the SE module is skipped and the scale remains unchanged.

(e) Frame 120, with the Max update strategy (top), the scale is not corrected because the candidate at unchanged scale is punished least. With the CWS update strategy (bottom), execution of the SE module has been continuously skipped.

(f) Frame 146, both update strategies terminate without adjusting the scale again.

Figure 5.14: The size plots and key frames of the TB100 *Dancer2* sequence. Better results are achieved by the CWS update strategy, because it prevents most of the shrinking out the bounding box that occurs with the Max update strategy. The ground truth bounding box is annotated in yellow and the predicted bounding box in magenta.

Even though this is a rather small error, this error accumulates over multiple sequences and is likely to be a strong contributor as to why the Candidates algorithm with the CWS update strategy achieves better results than the Candidates algorithm with the Max updates strategy. The fact that with the CWS update strategy the Candidates algorithm often fails to accurately adjust the scale seems to be less of a negative effect than the problem of the initial shrinking of the bounding box with the Max update strategy.

In accordance with the behavior of the DSST algorithm in combination with the CWS update strategy, the CWS has the same effect on the Candidates algorithm, where the algorithm becomes a lot less responsive towards changes in scale. This is because the algorithm gets executed on fewer frames and as a consequence simply has fewer chances to adapt the scale (but also fewer chances to output a wrong scale). This effect is shown in Figure 5.15, where the Candidates algorithm in combination with the CWS update strategy fails to accurately estimate the scale of the object, while Candidates algorithm in combination with the Max update strategy shows desirable scale estimation results.

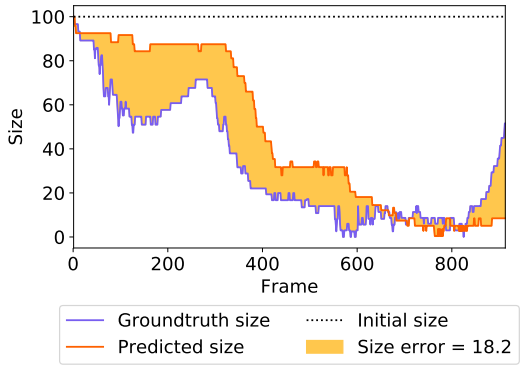
Summarizing the section, the DSST algorithm has shown to be capable of accurately estimating the scale, while the Candidates algorithm achieves best results in combination with the CWS update strategy, which limits the algorithms capabilities to decrease the objects scale, specifically at the beginning of the sequences. Further, for both algorithms, the CWS update strategy has the potential to prevent the SE module from execution on bad frames, but also has the downside of resulting in a less accurate scale estimation, because small scale changes are promptly not apprehended, due to the low resolution of the prediction mask.

5.6.2 Performance on NICO sequences

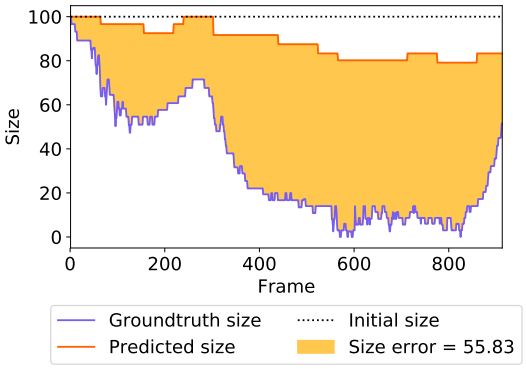
The NICO dataset contains exclusively sequences that show the NICO robot involved in interaction with various objects. Thus, most sequences show the same set of challenging aspects. For example, during the interaction, the NICO robot tends to occlude the target object with its hands. The occlusion is differently severe because the NICO robot shows different techniques for grasping, pushing or pulling the object, but partial occlusion has a strong presence in the dataset. Similarly common is a very sudden, strong change in the size of the object, as the actions of the NICO robot abruptly displace the object, often either closer (pulling or lifting actions) or further away (pushing actions) from the recording eye camera.

From the previous section, we already know that the CWS update strategy is helpful in dealing with occlusion, for multiple reasons. Firstly, the execution of the SE module is entirely skipped when HIOBs prediction quality for the final prediction on a frame, is too low. Secondly, the remainder of the object that is still visible and not occluded can still be good enough for tracking, in which case the execution of the SE module is skipped as well.

Under the consideration of the above, it is expected, that CWS update strategy achieves significantly better results in combination with the Candidates algorithm on the NICO dataset. The Candidates algorithm in combination with the Max up-



(a) The size plot of the Candidates algorithm with the max update strategy and dynamic aspect ratio implementation.



(b) The size plot of the Candidates aspect ratio with the CWS update strategy and dynamic aspect ratio implementation.

(c) Frame 129, with the max update strategy (top), the size has been decreased, while with the CWS update strategy (bottom), the scale is still almost at the initial size.

(d) Frame 412, with the max update strategy (top), the size of the bounding box has been further decreased, while with the CWS update strategy (bottom), the scale adjustment has been skipped a lot and is still close to the initial size.

(e) Frame 627, the max update strategy (top), maintains a close estimation of the scale.The CWS update strategy (bottom), prevents updating the scale because the visual features are still considered good enough.

(f) Frame 807, the max update strategy (top) is close to the ground truth bounding box. Even though the SE module is executed with the CWS update strategy (bottom), but it fails to reduce the scale.

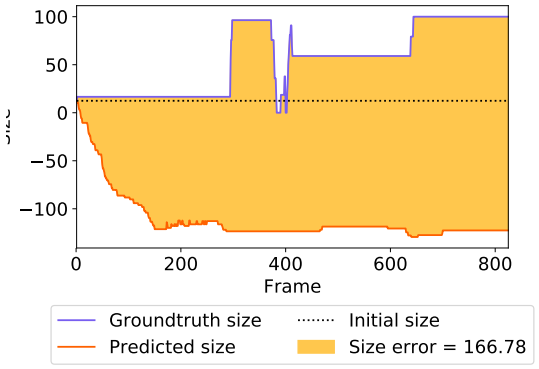
Figure 5.15: The size plots and key frames of the TB100 *Car2* sequence, showing that acceptable results can be achieved with the Max update strategy. Better results are achieved by the Max update strategy, because the scale estimation module is executed on every frame and is thus given more chances to adjust the scale. The ground truth bounding box is annotated in yellow and the predicted bounding box in magenta.

date strategy is prone to poor performance under occlusion because if the algorithm is executed during ongoing occlusion, the bounding box will be decreased to fit the size of the remainder of the object. Before we investigate exemplary sequences, it should be noted that the NICO dataset is recorded in high resolution, which means that the size score on this dataset has higher absolute values. Figure 5.16 gives an example on the NICO *lift_red_car_01* sequence, where the occlusion of the target object leads to a strong decrease in size, if the Max update strategy is used. The by now well-known problem of an initial decrease in size with the Candidates algorithm and the Max update strategy can be observed as well.

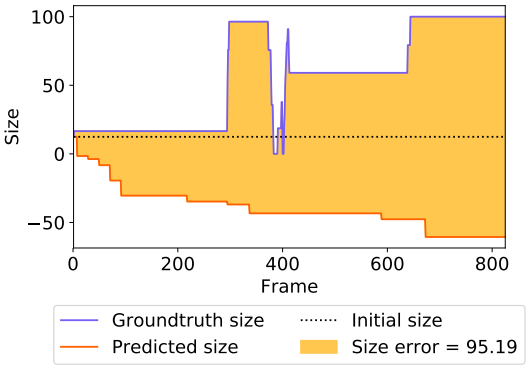
The results obtained by the Candidates algorithm in combination with the CWS update strategy on the NICO dataset are significantly better than the results from the Candidates algorithm with the Max update strategy. This is because the CWS update strategy suppresses the unintended shrinking of the bounding box which occurs when the Candidate algorithm is used in combination with the Max update strategy. This is in accordance with the findings from the TB100 dataset. However, the Candidates algorithm in combination with the CWS update strategy still appears to be incapable of correctly estimating the scale on the NICO dataset. The example provided in Figure 5.16 can be considered relatively representative for the other sequences from the NICO dataset. Generally, better results are achieved by the CWS update strategy with the Candidates algorithm, but even the better results are the result of preventing bad SE results instead of achieving good SE results.

The poor performance of the Candidates algorithm on the NICO dataset can be explained by multiple factors. The Candidates algorithm works on the prediction map produced by the visual features that are extracted by the VGG16 CNN. Along with strong occlusion, rotation of the object during interaction is another condition that occurs frequently in the dataset, which is also difficult for the CNN to handled and poses a small research field on its own. Finally, many sequences of the NICO dataset show a peak in the center distance during the interaction with the object, which indicates that during the interaction, HIOB sometimes tracks the hand of the robot instead of the actual object. Under consideration of those conditions, the poor performance of the Candidates algorithm, which operates based on the values on the prediction which is the output from the VGG16 CNN, can to a certain degree be justified.

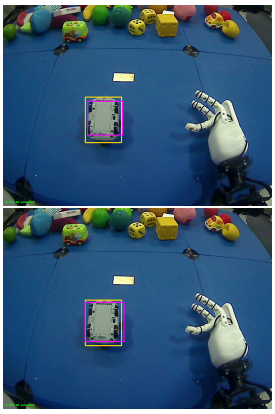
The DSST algorithm achieves consistent results on the NICO dataset with relatively small variance in the obtained success ratings, independently of the update strategy. The previous findings which indicate that the CWS update strategy reduces the overall variance of the SE module at the cost of slightly less accurate estimation can partly be confirmed on this dataset aswell, which is shown in Figure 5.16. However, while the results in Figure 5.17 show promising results, for the fast majority of sequences on the NICO dataset, the DSST algorithm does not manage to react to the very sudden and quick scale changes introduced by the NICO robot interaction with the target objects, which explains why the overall results achieved by the DSST algorithm are only slightly higher than the overall results of the Candidates algorithm and the baseline condition (which does not use



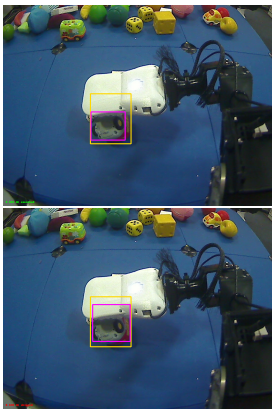
(a) The size plot of the Candidates algorithm with the Max update strategy and dynamic aspect ratio implementation.



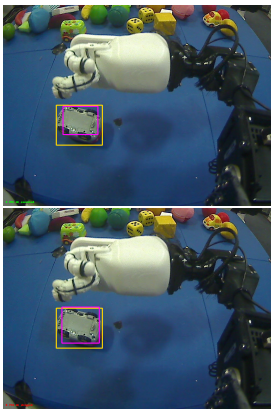
(b) The size plot of the Candidates algorithm with dynamic aspect ratio with the CWS update strategy and dynamic aspect ratio implementation.



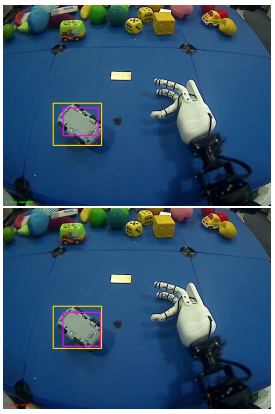
(c) Frame 68, with the max update strategy (top), the bounding box has been decreased to contain only what has a very high likelihood of belong to the object, while with the CWS update strategy (bottom) did not significantly decrease the scale.



(d) Frame 335, with the Max update strategy (top), the bounding box is further decreased to the remainder of the object, while with the CWS update strategy (bottom), SE execution is skipped.



(e) Frame 472, the Max update strategy (top) maintains the strongly decreased scale. The CWS update strategy (bottom) never decreased the scale as drastically and still achieves relatively high overlap.

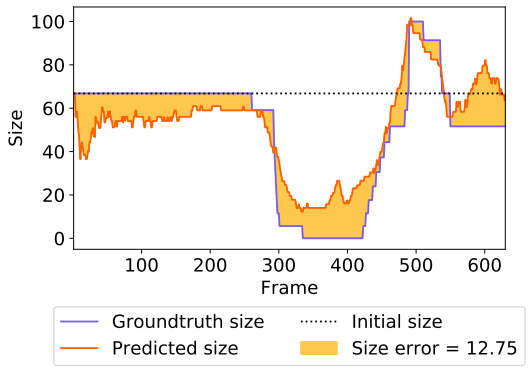


(f) Frame 713, without further adjustments to the scale, both algorithms terminate with an overall too small bounding box.

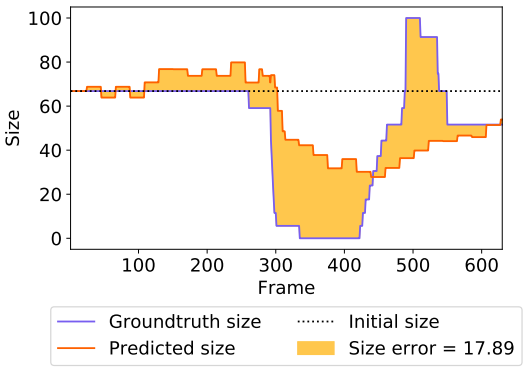
Figure 5.16: The size plots and key frames of the NICO *lift_red_car1* sequence. The jump in scale between frame 200 and 400 is caused by initial lifting, follows by dropping of the target object. Better results are achieved by the CWS update strategy, because the CWS update strategy skipped the execution of the SE module under occlusion. The ground truth bounding box is annotated in yellow and the predicted bounding box in magenta.

the SE module). Instead, the results on the NICO *push_blue_ball* are likely to as good because only very few to none occlusion takes places during the interaction of the NICO robot with the ball, which enables the algorithm to compute the scale factors without significant problems.

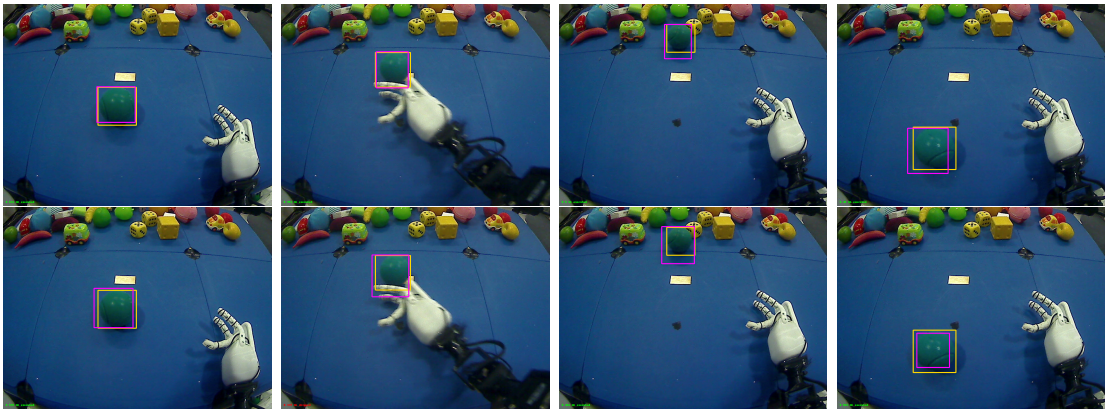
Summarizing this section, the general relationships and properties of the algorithms and their combination with the update strategies are the same as on the TB100 dataset. The Max update strategy in combination with the Candidates algorithm decreases the scale too much, which happens because the outer borders of the object, the likelihoods on the prediction mask are smaller than at the inner area. The Candidates algorithm with the CWS update strategy achieves better results than when the Max update strategy is employed, because the CWS update strategy limits the described unintended decrease in size, but does not accurately estimate the scale either. The DSST algorithm achieves good results on a small amount of the NICO sequences but fails to capture the very abrupt scale changes introduced in the robot interaction environment.



(a) The size plot of the DSST algorithm with the Max update strategy and dynamic aspect ratio implementation.



(b) The size plot of the DSST algorithmwith the HGC update strategy and dynamic aspect ratio implementation.



(c) Frame 218, both the Max update strategy (top) and the CWS update strategy (bottom) shows bounding boxes close to the ground truth. Representation updates have been trigger by occasional, slight shadows being cast on the canvas.

(d) Frame 293, both update strategies maintain similar bounding boxes, while with the CWS update strategy (bottom), execution of the SE module has been skipped during robot interaction, resulting in a slightly bigger bounding box.

(e) Frame 322, the Max update strategy (top) adjusted to the scale decrease. The CWS update strategy (bottom) fails to adjust the scale, possible because the recent predictions were not properly centered on the ball.

(f) Frame 493, the Max update strategy (top) adjusted to the returning ball, while the CWS update strategy (bottom) still reacts to the returning ball by increasing the scale.

Figure 5.17: The size plots and key frames of the NICO *push_blue_ball* sequence. Slightly better results are achieved by the Max update strategy, because faster adjustments can be made. The ground truth bounding box is annotated in yellow and the predicted bounding box in magenta.

Chapter 6

Conclusions

The main goal of this thesis is to provide a solution to the problem of scale estimation in visual object tracking. For that, an analysis of the available algorithms in state-of-the-art trackers has been conducted and two algorithms have been selected, implemented, and thoroughly tested.

6.1 Summary of Contributions

The main contribution of this thesis is the implementation of a new module in the HIOB tracking framework, which is capable of accurately estimating the scale of the target object during tracking. To achieve this goal, two separate algorithms have been implemented, namely the *scaled candidates* algorithm and the DSST scale estimation algorithm by Danelljan et al. [9]. Both algorithms have been extended to feature independent scaling on the x and y-axis and can be configured to use different update strategies.

6.1.1 Scale estimation algorithms

The first algorithm that has been implemented in the SE module, is dubbed the *scaled candidates* algorithm. The algorithm has been proposed by Peer Springstübe as an extension to the HIOB tracking framework [30]. The scaled candidates algorithm has been extended to be capable of handling slight rotations and transformations of the target object, by maintaining two separate scale factors, one factor for the x and one for the y-axis. This feature has been shown to consistently achieve slightly better results than the one-factor counterpart.

The second algorithm that has been implemented is referred to as the DSST algorithm throughout this thesis and has been introduced by Danelljan et al. [8, 9]. In accordance with the work on the Candidates algorithm, the DSST algorithm has also been extended to feature the independent scaling on the x and y-axis.

In his original work on the HIOB tracking framework, Peer Springstübe found that specific update strategies can prevent HIOBs internal representation model from being trained on bad frames. Motivated by this finding, the best performing

update strategy has been adapted and the two scale estimation algorithms are configurable to either use the *Confidence window strategy* or be executed on every frame.

6.1.2 The size error metric

To obtain a specific measurement for the performance of the SE module, a new metric has been introduced, dubbed the size error. The size error obtains a numerical rating for how similar the sizes of the predicted bounding box and the ground truth bounding box are, by accumulating the difference in the products of the x and y-axis values for each frame. This is a different measurement as the success metric because the success metric measures the overlap between the predicted bounding box and the ground truth bounding box.

With the size error, the behavior of the different versions of the two main algorithms has been analyzed, which revealed how the update strategies affect the scale estimation algorithms. The size error has also underlined the difference between the scale variation challenges on the TB100 dataset in comparison to those present on the NICO dataset.

6.2 Discussion

Throughout this thesis, insight has been gained regarding the performance of the two SE algorithms and a clear trend regarding the update strategies has been identified. The DSST algorithm has been proven to be capable of accurately estimating the scale of arbitrary objects in various tracking sequences from the TB100. Intended behavior of the algorithm has been confirmed by an extra experiment, comparing the results achieved of the reference DSST tracker and the HIOB tracker with the SE module. For this, the SE module has been configured to use the DSST algorithm with the static bounding box implementation and to execute the algorithm on every frame. While the overall results of the HIOB tracker were not as good as the results achieved by Danelljan et al.'s DSST tracker, the implementation of the algorithm in HIOB still shows strong scale estimation results, and the results could partly be explained by identifying outliers in the results that were caused by misplaced predictions. Consistently slightly less accurate estimation of the scale with the HIOB implementation of the algorithm has been identified and is likely to be caused by different feature extractor implementations.

The scaled candidates algorithm with its current parameter configuration does not achieve satisfying scale estimation results. Although the algorithm has shown accurate scale estimation results on a few sequences from the TB100 dataset, even strong scale variations under favorable conditions are not always correctly handled by the algorithm. The fact that the algorithm does show good results on a few sequences however indicates, that the configuration obtained from the optimization experiment might not be ideal, as the behavior of the algorithm can be explained by the parameter values obtained from the configuration.

Regarding the update strategies, the *confidence window strategy* shows to behave as expected, meaning the update strategy prevented the execution of the SE module on frames that could potentially lead to the production of bad scale factors. This has been shown to greatly benefit the scaled candidates algorithm. The DSST algorithm does not benefit from the CWS as much as the scaled candidates algorithm does, however, when facing the challenging aspects from the NICO dataset, the DSST algorithm achieved best results with the confidence window update strategy. This allows the conclusion, that the HGC update strategy is applicable to scale estimation, but also results in a slightly broader approximation of the scale.

Finally, the extension of the two algorithms which enables a dynamic aspect ration has been shown to be partly beneficial. On the TB100, both algorithms achieve best results when the dynamic aspect ratio implementation is used. However, the increase in the success rating is marginal, and specifically for the DSST algorithm, the additional computational load is not justified by such a small increase in the success rating.

The sequences of the NICO dataset have shown to be very challenging for both algorithms. Even the DSST algorithm that shows strong scale estimation results on many sequences of the TB100 dataset does not manage to successfully handle the abrupt, fast, and partially occluded scale changes that are present on the NICO dataset.

Considering the research objectives, with the DSST algorithm a state-of-the-art algorithm has been found that is applicable to the HIOB framework and its combination with the NICO robot. The computational load of the static version of the algorithm has been found, when executed on every frame, to decrease HIOBs frame rate by 20%. However, when the confidence window update strategy is employed, the DSST algorithm does not result in a significant decrease in processing speed. This satisfies the main constraints of the developmental robotics, which is the necessity of operating in real time. Further, the main challenge of scale estimation has identified as the need to adapt the internal model representation to the changing scale. In the HIOB framework, this has been achieved by implementing the scale estimation module, so that the scale adapted object representation can be learned online.

6.3 Future Work

Evaluating the results achieved by the scaled candidates algorithm on the TB100 and NICO dataset revealed multiple problems with the configuration that has been obtained by optimizing the scaled candidate parameters independently of one another. Changing the scale is punished by a large factor, which explains why the algorithm failed to often accurately adapt to the scale changes of different objects. Additionally, the algorithm tends to decrease the scale even when the object is static and not decreasing its size because the values on the outer borders of the object have a lower likelihood of belonging to the object and thus, candidates are

punished for containing the borders. This behavior is controllable by the values of the *Inner_punish_threshold* and *Outer_punish_threshold* parameters and could be prevented. As described in subsection 5.1.4, the baseline configuration used for optimization might not have been ideal, which explains why the best results were obtained when changing the scale was strongly punished. Thus, further investigation regarding the parameter setting for the scaled candidates algorithm is likely to yield better results.

Additionally, in the DSST validation experiment, it has been revealed that the DSST algorithm as implemented in HIOB consistently achieves slightly less accurate scale estimation results than the reference implementation by Danelljan et al [9]. It is likely that this occurs because of implementation details in the HOG features descriptor, Investigating this would be interesting because it is possible that with a better feature descriptor, the algorithm might be capable of handling the scale variations on the NICO dataset.

Danelljan et al. propose different strategies to reduce the cost associated with the DSST algorithm that have not been implemented in this thesis. Building upon the promising scale estimation results shown by the algorithm, those could be implemented to further lower the computational cost of the algorithm.

As the main challenges on the NICO dataset have been identified to be the combination of occlusion and fast scale changes, the texture of the target could be extracted using local binary patterns and maintained as an additional feature channel (next to HOG), to help with the partial occlusion from NICOs hands [26]. Further, instead of using a Hann window to smooth the scale output, a momentum strategy could be implemented to cope with the sudden scale variation of the NICO dataset.

This thesis provides a viable solution for scale estimation in a CNN based tracking context. By employing a specific metric, deep understanding of the behavior of the different scale estimation algorithms has been obtained and it could be shown how the use of an update strategy can help to produce a more stable estimation of the scale.

Bibliography

[1] Ning An, Shi-Ying Sun, Xiao-Guang Zhao, and Zeng-Guang Hou. Remember like humans: Visual tracking with cognitive psychological memory model. *International Journal of Advanced Robotic Systems*, 14(1), 2017.

[2] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1619–1632, Aug 2011.

[3] David. S Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui. M. Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, June 2010.

[4] David. S. Bolme, Bruce. A. Draper, and J. Ross Beveridge. Average of synthetic exact filters. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2105–2112, June 2009.

[5] Ronald Newbold Bracewell. *The Fourier transform and its applications; 2nd ed.* McGraw-Hill series in electrical engineering. Circuits and systems. McGraw-Hill, New York, NY, 1986.

[6] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *International Conference on Computer Vision & Pattern Recognition (CVPR '05)*, volume 1, pages 886–893, San Diego, United States, June 2005. IEEE Computer Society.

[7] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg. Dsst tracker reference implementation. `http://www.cvl.isy.liu.se/research/objrec/visualtracking/scalvistrack/index.html`, 2017. [Online; accessed 8-April-2019].

[8] Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *Proceedings of the British Machine Vision Conference 2014 :*. BMVA Press, 2014.

[9] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Discriminative scale space tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1561–1575, Aug 2017.

[10] Piotr Dollár. Piotr's Computer Vision Matlab Toolbox (PMT). `https://github.com/pdollar/toolbox`. [Online; accessed 8-April-2019].

[11] Qianyun Du, Zhao-quan Cai, Hao Liu, and Zhu Liang Yu. A rotation adaptive correlation filter for robust tracking. In *2015 IEEE International Conference on Digital Signal Processing (DSP)*, pages 1035–1038, July 2015.

[12] Richard O. Duda and Peter E. Hart. *Pattern classification and scene analysis*. 1973.

[13] Pedro Felzenszwalb, Ross Girshick, David McAllester, and Deva Ramanan. Visual object detection with deformable part models. *Commun. ACM*, 56(9):97–105, September 2013.

[14] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[16] Sion Hannuna, Massimo Camplani, Jake Hall, Majid Mirmehdi, Dima Damen, Tilo Burghardt, Adeline Paiement, and Lili Tao. Ds-kcf: a real-time tracker for rgb-d data. *Journal of Real-Time Image Processing*, Nov 2016.

[17] Stefan Heinrich, Peer Springstübe, Tobias Knöppler, Matthias Kerzel, and Stefan Wermter. Continuous convolutional object tracking in developmental robot scenarios. *Neurocomputing*, page 8, Feb 2019.

[18] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, July 2012.

[19] Matthias Kerzel, Erik Strahl, Sven Magg, Nicolás Navarro-Guerrero, Stefan Heinrich, and Stefan Wermter. Nico - neuro-inspired companion: A developmental humanoid robot platform for multimodal interaction. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 113–120, Aug 2017.

[20] Reinhard Klette. *Concise Computer Vision: An Introduction into Theory and Algorithms*. Springer Publishing Company, Incorporated, 2014.

[21] Tobias Knöppler. Visual object tracking for robotic applications. Bachelor's thesis, University of Hamburg, dept. Knowledge Technology, 2017.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[23] Yann LeCun, Y Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[24] Yann Lecun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS 1989), Denver, CO*, volume 2. Morgan Kaufmann, 1990.

[25] Yang Li and Jianke Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Computer Vision - ECCV 2014 Workshops*, pages 254–265, Cham, 2015. Springer International Publishing.

[26] Kourosh Meshgi, Maeda. Shin-ichi, Shigeyuki Oba, Henrik Skibbe, Yu-zhe Li, and Shin Ishii. An occlusion-aware particle filter tracker to handle complex and persistent occlusions. *Computer Vision and Image Understanding*, 150:81 – 94, 2016.

[27] Nayyab Naseem and Mehreen Sirshar. Target tracking in real time surveillance cameras and videos. *CoRR*, abs/1506.06659, 2015.

[28] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1):125–141, May 2008.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[30] Peer Springstübe. Object tracking with convolutional neural networks. Diploma thesis, University of Hamburg, dept. Knowledge Technology, 2017.

[31] Chong Sun, Dong Wang, Huchuan Lu, and Ming-Hsuan Yang. Learning spatial-aware regressions for visual tracking. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[32] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[33] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[34] Yulong Xu, Jiabao Wang, Hang Li, Yang Li, Zhuang Miao, and Yafei Zhang. Patch-based scale calculation for real-time visual tracking. *IEEE Signal Processing Letters*, 23(1):40–44, Jan 2016.

[35] Hanxuan Yang, Ling Shao, Feng Zheng, Liang Wang, and Zhan Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823 – 3831, 2011.

Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Bachelorthesis im Studiengang Human-Computer Interaction selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Bachelorthesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift