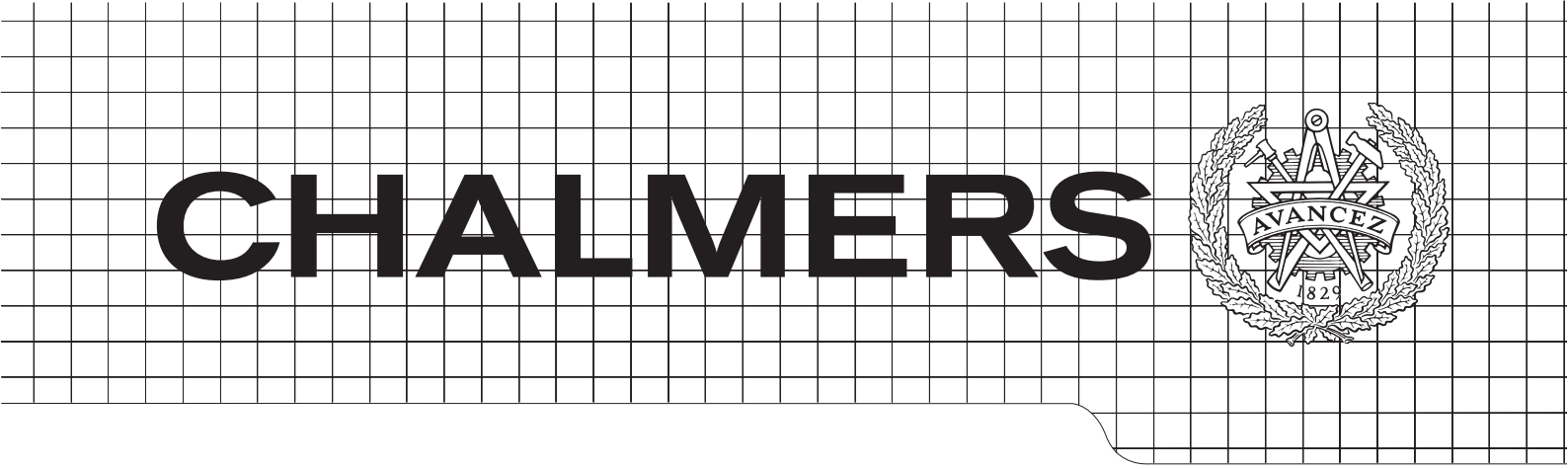


Evaluation of Background Subtraction in Pan-Tilt Camera Tracking

Master's thesis in Complex Adaptive Systems

MARKUS HÄGERSTRAND
HJALMAR KARLSSON

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2016
Master's Thesis EX040/2016

Master's Thesis EX040/2016

Evaluation of Background Subtraction in Pan-Tilt Camera Tracking

Markus Hägerstrand
Hjalmar Karlsson

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

Evaluation of Background Subtraction in Pan-Tilt Camera Tracking

Markus Hägerstrand
Hjalmar Karlsson

Supervisor: Harald Freij, SAAB AB
Examiner: Fredrik Kahl, Department of Signals & Systems, Chalmers

Master's Thesis EX040/2016

Department of Signals & Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46(0)31-772 1000

Gothenburg, Sweden 2016

Evaluation of Background Subtraction in Pan-Tilt Camera Tracking

Markus Hägerstrand
Hjalmar Karlsson
Department of Signals & Systems
Chalmers University of Technology

Abstract

Object tracking is the subfield of computer vision where an object is to be located in each frame of a video sequence. Automated tracking is useful in all areas where vision and cameras are used. Computers can assist in time-consuming tasks in television or surveillance as well as stabilise and increase tracking precision compared to manual operation. In a system using a movable camera such as a pan-tilt-zoom camera mounted on a robot, information about the pan-tilt-zoom configuration can be used to locate a moving object in successive frames since static background can be accounted for from one frame to the next. Two state-of-the-art trackers, called Adaptive Scale Mean Shift (ASMS) and Kernel Correlation Filter (KCF), as well as a tracker based on Shi-Tomasi corner detection together with optical flow (OPTFLOW), are in this thesis evaluated when using a pre-processing stage of online background subtraction based on the historical pixel value distribution. All trackers with and without background subtraction were evaluated for robustness on multiple scenarios containing either a circular unicoloured object or a multicoloured polygon in front of two different backgrounds respectively. The tracking performance was shown to not benefit from this particular background subtraction since the amount of wrongly classified background pixels ruined more than the correctly classified pixels helped. The implemented background subtraction model affected OPTFLOW the most since the background subtraction removed important corner features, while ASMS and KCF were robust and unaffected by the background subtraction. The background subtraction routine for a static camera view was successfully adapted to function for a translating camera, and may be of more use for some trackers not evaluated.

Keywords: Tracking, Background subtraction, PTZ

Acknowledgements

This thesis work has been conducted at SAAB AB located in Gothenburg. We would like to express our great appreciation to the supervisor Harald Freij for guiding us throughout the project. Also, we are grateful for the valuable input on object tracking given by Martin Lillieborg and Henrik Söderström. Finally, we wish to thank our friends and family for their support.

Markus Hägerstrand & Hjalmar Karlsson, Gothenburg, June 13, 2016

Contents

Glossary

ASMS Adaptive Scale Mean Shift (tracker)

BGR Blue-Green-Red, colour space

BRW Background Ratio Weighting

DFT Discrete Fourier Transform

Egomotion Motion of the camera

HOG Histogram of Oriented Gradients

KCF Kernelized Correlation Filter (tracker)

KDE Kernel Density Estimation

MS Mean Shift

OpenCV Computer Vision library

OPTFLOW The optical flow tracker

PTZ Pan-tilt-zoom

VOT Visual Object Tracking

1

Introduction

Visual surveillance systems are in use in many places, the applications range from passive monitoring to active use as an aid for operators, e.g. increase their field of vision. Automatic tracking of objects of interest decreases the need of human supervision of the camera.

The task of tracking can generally be described as two steps, detection of objects and updating the position of the objects in consecutive frames in a video sequence. The focus of this thesis is on pan-tilt-zoom (PTZ) cameras, movable cameras restricted to pan, tilt and zoom motions. This type of camera is suitable for keeping a single object in focus, and thus only tracking of a single object at a time is studied. As only a single object is to be tracked, the initialisation of the object of interest is done by the operator. It is possible to construct a system that performs object detection and presents a list of objects for an operator to choose from. Since that system also would require human intervention, the system studied in this thesis is designed so that the initial detection is done manually by selecting a region that contains only a single object.

Even if the initial detection is performed manually the next step of the tracking (updating the position of the object) can benefit from some form of object detection, segmenting foreground and background objects. By filtering out the background the complexity of the image is reduced which may help with the tracking. For stationary cameras the task of segmenting background and foreground for detecting objects of interest is well-studied. For freely moving cameras or cameras with restricted motion this is not the case.

Pre-processing videos by segmenting background or foreground has been done before [1, 2, 3]. In particular [2] presents a framework for performing background subtraction with data from a PTZ camera by using recent frames and known per frame offset to build a model for each pixel. A common approach to get knowledge of the camera motion is through keypoint tracking, which is what is done in [2].

When the control of the PTZ is integrated with the tracker the computation of the

camera egomotion can be simplified, as each frame comes with logged information about the pan and tilt angles. The rotation of the camera causes a different image to get projected onto the image plane, when the rotation is known this can be compensated for [4, 5]. Performing this transformation is computationally cheaper than matching keypoints.

To keep track of the detected objects many different algorithms are available [6], and which to choose depends on a number of factors. Important factors for applications in surveillance are speed and robustness. That means the tracker should be able to work in real time, i.e. at least 30 frames per second, and should not lose track of the object too often.

The majority of research in computer vision with focus on tracking tries to solve the very general problem: given a video stream from an unrestricted camera, keep track of the motion of some object(s). By restricting the motion of the camera to changes in pan, tilt and zoom some simplifications are possible. Performing background subtraction is intractable in the general case unless very detailed information about the movement is known, and usually no such data exist and methods such as matching keypoints has to be employed. Those methods come with their own problems, is the movement of the keypoint from camera egomotion or did the object move?

Performing background subtraction for tracking has been done before [5, 7], what is new in this work is evaluating the impact of using background subtraction before tracking.

To compare the performance between trackers there are frameworks where trackers are evaluated on a diverse dataset to obtain measures of accuracy and robustness as well as a ranking relative to other trackers. One such framework is the VOT Challenge [8].

1.1 Review of motion tracking

The first step of the tracking, object detection, can be done in many ways. How to do it depends on data available and whether the object is in motion or not. For objects at rest some prior knowledge regarding the type of objects must be known. This can be a single sample image of the object to track. Detecting moving objects in an image sequence does not need prior knowledge but needs multiple consecutive images. Two common methods for detecting moving objects are [9]:

Background subtraction: If the view of the camera is known it is possible to perform background subtraction. The principle is that if a reference background image is known, that image can be compared with the frame in which objects are to be detected. The regions that are different contain moving objects.

Optical flow: By calculating the flow field of pixels in successive frames it is possible to detect objects. Clusters of pixels moving together are likely to be part of the same object.

When the location of the object to be tracked is known some features must be extracted and recorded to make it possible to find the same object in new frames. Good segmen-

tation from the background ensures that only features that actually belong to the object of interest are recorded. The problem is thus, given an area containing an object, to determine which pixels belong to the object and which belong to the background. In some cases a pixel-wise segmentation is not needed, but if too much background gets incorporated in the object model the noise will make it very hard to keep track of the target.

Objects can be represented in multiple ways, as a centroid point, multiple points, primitive geometric shapes or object contours and silhouettes. These can be combined to get a good representation of the object that is to be tracked. Good features to track are things that continue looking the same even if the scale changes or the object rotates out of plane. Examples of that kind of features are corners and edges. Another possible representation of the object is the colour histogram of the object area [10].

When features have been extracted they need to be tracked. How this is done depends to some extent on the features used. Two broad categories of tracking algorithms are [6]:

Point tracking: A detector has selected some points that are located on an object, these points are then tracked from frame to frame based on assumptions about how much a given point can move and change between frames.

Kernel tracking: Kernel refers to the object shape and appearance. Usually this is some bounding box for the object and a histogram. Objects are tracked by computing the motion of the kernel.

Point tracking can be done by calculating the optical flow between images and keeping track of the locations of interesting features, or Kalman filters can be used to predict the motion of the feature points [11, 12]. Point based tracking performs recognition (and keeps track of whether tracking is lost or not) by clustering the extracted points into higher level features that can be matched between frames [9].

1.2 Background subtraction

In the most basic sense background subtraction is nothing more than what the name implies, the absolute difference between a reference image (the background) and an image of interest. At image positions where the difference is greater than some threshold the position is classified as not belonging to the background, i.e. classified as a foreground pixel. Most modern algorithms for performing background subtraction are more complex than this and can be divided into a couple of categories. The main difference between most methods is how the background model is represented. From simple to more complex ones:

Running Gaussian Average: For each pixel the background is modelled independently as a Gaussian probability density function. The Gaussian distribution is fitted to the n latest pixel values and a pixel is classified by calculating the probability that the latest pixel value describes the same object as the earlier pixel values did.

Mixture of Gaussians: Sometimes the part of an image that should be classified as background is not entirely static, some parts might move a little (due to wind, vibrations of the camera etc.) and should still be classified as background. To cope with that kind of background a single-valued background model is insufficient. The idea is to have different Gaussian models for different possible background objects, if a pixel value is unlikely to come from any of the different distributions then it is classified as foreground.

Kernel density estimation (KDE): In this method a function is constructed that gives the probability that a given pixel belongs to the distribution of background pixels. For the Gaussian running average the previous known pixel values were fitted to a Gaussian to model the distribution, in the kernel density estimator the distribution is instead constructed from a sum of kernels.

For a more detailed review of different methods and how they differ see: [13, 14]. A variant of kernel density estimation for background subtraction is implemented in [15] and is explained in more detail in section 2.1. An example of an implementation of a Gaussian Mixture Model for background subtraction with a PTZ camera is described in [2].

Another approach to background subtraction is to calculate a partial optical flow for the video sequence and classifying different point trajectories as either belonging to the foreground or the background based on constraints on the movements [1].

1.3 Purpose and scope

The goal of this thesis is to study the impact of background subtraction on tracking in the scenario that the camera is restricted to pan-tilt motions. Restriction of the motion reduces the complexity of matching pixel locations between subsequent frames, prior knowledge of the changes in pan and tilt angle further reduces the complexity as the camera egomotion does not have to be computed. Tracking is only done at fixed zoom levels with no change in zoom level during tracking. Allowing zoom would require some means to rescale the bounding box for the trackers and updating the model, while this can be done, it is outside the scope of this thesis.

To achieve the goal a background subtractor must be implemented, as well as finding trackers that are able to track both with and without the background subtraction. The main measures which will be used for the evaluation will be computational performance (frames processed per second) and robustness. Three different trackers that meet the requirements of speed and robustness in tracking without background subtraction are selected and evaluated with background subtraction.

The trackers are expected to be able to process new frames at minimum 30 fps given that the resolution of the video stream is not higher than 720×480 pixels. The system will only be evaluated at fixed zoom levels, no zooming will be done during a sequence and the tracked objects are assumed to be rigid. In addition it is assumed that motion

between successive frames (change in pan and tilt of the camera) is small enough that it can be approximated as a pure translation of the scene.

The three trackers chosen are "Adaptive Scale Mean Shift" (ASMS) [16], "Kernelized Correlation Filter" (KCF) [17], both modified from publicly available code[1,2], and a tracker we call OPTFLOW. These three track on different features, ASMS represents the target with a colour histogram, KCF calculates a histogram of gradients that is used to create a template and OPTFLOW tracks feature points (corners).

1.4 Outline of the report

There are five chapters in this report: Introduction, Theory, Method, Results, and Discussion. This first chapter introduced the concepts of tracking and background subtraction as well as the motivation and goal of the thesis. In the theory chapter the algorithms and some technical details from the background subtraction and the three trackers are presented, information later used in the discussion chapter. The method chapter consists of a system overview and what was carried out in the project together with the organisation of the test procedures. The results chapter presents the results of the tests described in the method chapter. The last chapter is the discussion about relevant aspects regarding tracking and background subtraction, based on the theory and results chapter.

[1] `https://github.com/vojirt/asms`
[2] `https://github.com/joaofaro/KCFcpp`

2

Theory

This chapter provides the theoretical background for the different tracking algorithms and the background subtraction. It also contains some notes on how to measure tracking performance.

2.1 Background subtraction

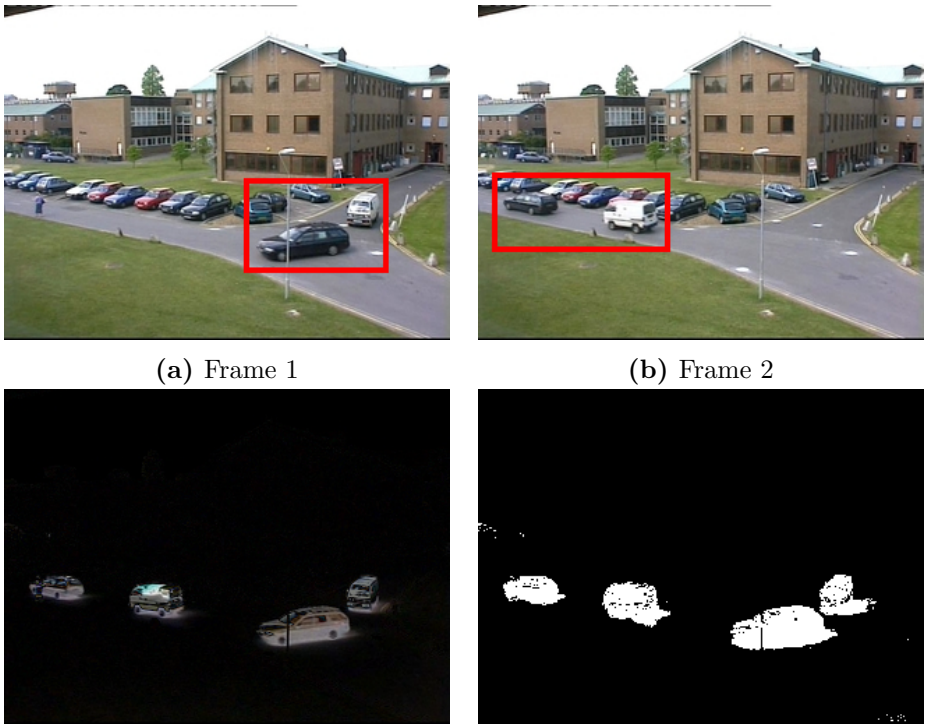
In the introduction there is an overview introducing different ways to perform background subtraction. The following section explains the algorithm implemented and used in this thesis. It is based on [15] with some modifications to make it suitable for use with a non-stationary camera. First an example demonstrating the simplest version of background subtraction.

A simple and intuitive way to perform background subtraction is to calculate the absolute difference between two consecutive frames. The images F_n and F_{n+1} are converted to greyscale ($F_n(x)$ is the colour intensity of the image point at x) and the subtraction produces a binary mask K where

$$K(x) = \begin{cases} 1 & \text{if } |F_n(x) - F_{n+1}(x)| > \tau \\ 0 & \text{else.} \end{cases} \tag{2.1}$$

The value 1 represents that the pixel belongs to the foreground and 0 that it is part of the background, τ is the threshold for how much difference in colour intensity that is allowed. An example of the result of doing this kind of background subtraction can be seen in figure 2.1.

A more sophisticated algorithm for background subtraction works by constructing a more complex model of the background. The background model is constructed from the N most recent frames, assumed known is the transformation from camera egomotion between consecutive frames. For an illustration of how the background model is



(a) Frame 1 **(b)** Frame 2

(c) Absolute difference **(d)** Thresholded

Figure 2.1: Example of the result of taking the absolute difference between two frames. In 2.1c the raw result is displayed and in 2.1d the result after setting all pixels with luminosity of over 20 (in any channel) to white and the rest black. Pixels are represented as colour intensities in the BGR channels with values from 0 to 255. Top figures from "car" sequence in the VOT2013 dataset [18].

constructed see figure 2.2. In the case with a PTZ camera this is calculated from the known absolute pan and tilt position for each frame.

A probability estimate for classifying pixels as background or foreground is calculated as:

$$Pr(x_t) = \frac{1}{n}\sum_{i=1}^{n}\prod_{j=1}^{d}\frac{1}{\sqrt{2\pi\sigma_j^2}}\exp\left[-\frac{1}{2}\frac{(x_{t_j}-x_{i_j})^2}{\sigma_j^2}\right]. \tag{2.2}$$

For each pixel x_t ($x_t = F_t(x)$) in the latest frame, n previous values are known ($n \leq N$ due to movement of the camera, for a stationary camera $n = N$). σ_j^2 is the kernel bandwidth for a given colour channel j, for a greyscale image $d = 1$ and for a colour image usually $d = 3$, e.g. blue-green-red (BGR). Equation (2.2) is equation (5) in [15].

The parameters that need to be set to segment the frame is thus a threshold τ such that if $Pr(x_t) > \tau$ then x_t is a background pixel, and the kernel bandwidths σ_j^2. The value of the kernel bandwidths limits the size of the pixel fluctuations that are filtered out. The value should be set to allow small variations due to noise in the image but

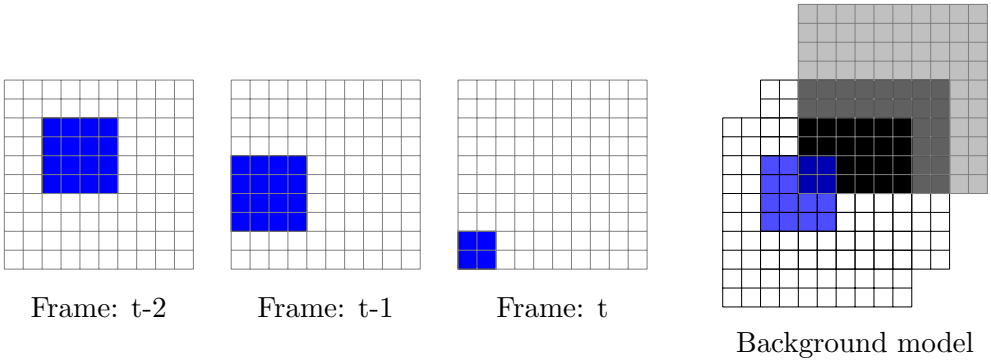


Figure 2.2: Background model constructed from three consecutive frames. Pixels in the black area have $n = 3$, pixels in the grey area have $n = 2$ and pixels in the light grey area $n = 1$. The other pixels are outside of the current frame and discarded. The blue rectangle is a stationary background object.

not large changes in pixel intensity value (from an object with different colour moving in front of something else). If a too high value is used only objects with a high contrast compared to the background will be detected, setting it too low results in many false foreground pixels. From [15] a good estimate is:

$$\sigma_j^2 = \frac{m_j^2}{0.68^2 \cdot 2} \tag{2.3}$$

were m_j is the median of $|x_i - x_{i+1}|$ for colour channel j with x going over all pixels and $i \in [0,n)$.

A major challenge is how to update the background model to get both good detection of moving objects and avoid incorrect detection of stationary objects (due to errors in the background model). Given a new pixel sample one can do a blind update and add it to the background model regardless of whether it has been classified as background or not. Otherwise one can choose to only add those pixels that have been classified as background.

By blindly updating the background model, pixels that are part of the foreground can accidentally get added to the model. This usually results in pixels in the centre of the moving object getting classified as background.

If updates to the background model are done by only adding pixels that have been classified as background on the other hand. One can end up in a deadlock where pixels that have once been classified incorrectly as background makes the real background get classified as foreground. An example of the difference can be seen in figure 2.3 from the sequence "car" from the VOT Challenge 2013 dataset[1] [18].

In order to get the best of both update models two separate background histories are recorded, one that is updated selectively, only adding values that are believed to be

[1] `http://votchallenge.net`

(a) Current frame

(b) Selective update result

(c) Blind update result

Figure 2.3: When selective updating is performed the moving object is segmented with very good precision but the position where the object start to move from in the first frame get initially classified as background, thus when the object moves away the real background gets classified as foreground. In the model performing blind update the segmentation is not as good but incorrect classifications does not persist. Top figure from "car" sequence in the VOT2013 dataset [18].

background and one that is updated blindly. Then, for pixels to be classified as foreground both models must classify them as foreground. Two separate histories introduce more parameters, determining how many frames (i.e. values for each pixel) should be kept in each model, and if all frames are included in each model or only every W:th frame [15].

To increase performance, in terms of suppression of false detections and decreasing processing time, the algorithm is modified to run on image pyramids. The pyramid is constructed by scaling down the original frame M times, then level 0 is the full scale image $[W \times H]$ and level l has been downscaled to $\frac{1}{2^l}[W \times H]$, see figure 2.4 for an illustration. Starting at level $l = M - 1$ the background subtraction is performed with a threshold τ_l. The result is then scaled up and used as a mask on level $l \leftarrow l - 1$. Only pixels classified as foreground on the coarser level is considered to be foreground candidates on the finer level.

This reduces the computational load as many pixels do not need to be evaluated on the full resolution image. Another benefit is suppression of false detections due to flicker

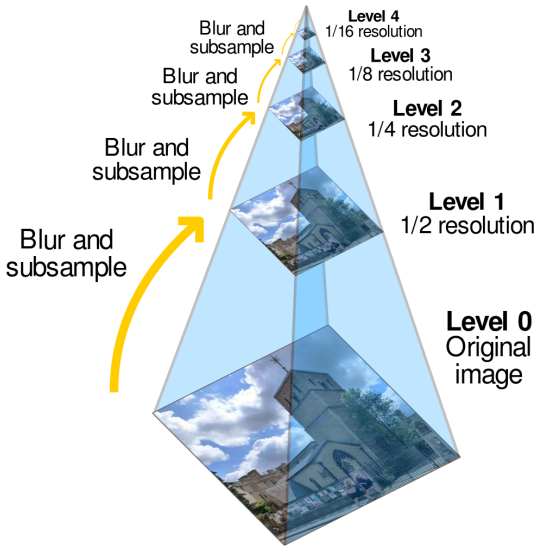


Figure 2.4: Illustration of image pyramids. Source: `https://en.wikipedia.org/wiki/File:Image_pyramid.svg`.

of single pixels as those changes get blurred and thresholded away in the coarser images.

A complete list of the parameters for the background subtraction:

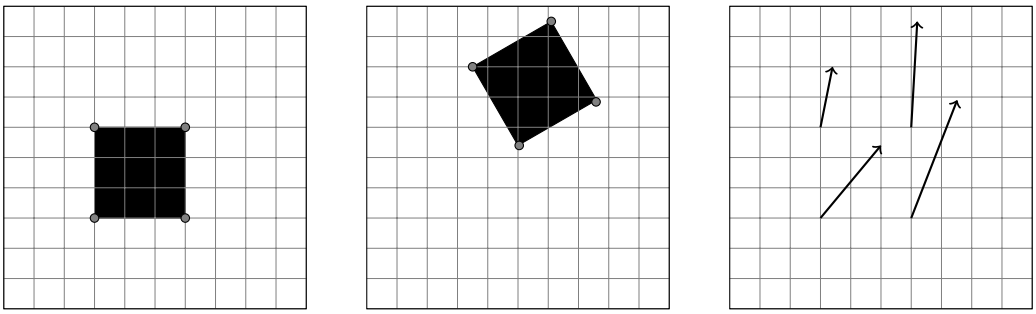
- σ_j^2, can either be set to a fixed value or estimated by the algorithm. It is the bandwidth of the kernel and can be seen as a value for how much the density estimate is smoothed.
- N_{lt}, number of frames in the background model that is blindly updated.
- N_{st}, number of frames in the selectively updated model.
- W, only every W:th frame is added to the blind update model.
- n_{pyr}, number of image pyramids.
- *threshold*, how high the background pixel probability must be for a pixel to be classified as background.

How these parameters are chosen depends on the movement of the tracked object. W and the size of the blind update history N_{lt} should be set so that incorrect updates in the selectively updated model are suppressed.

2.2 Optical flow

Optical flow describes the motion field for objects in an image. When it has been calculated it is possible to use it for image segmentation and tracking. The optical flow

problem can be formulated as follows: given an image point $\mathbf{u} = [u_x,u_y]^T$ in frame F_n, find $\mathbf{v} = [u_x+\delta_x,u_y+\delta_y]^T$ in F_{n+1} such that $F_n(\mathbf{u})$ and $F_{n+1}(\mathbf{v})$ are similar, see figure 2.5 for an example of the optical flow of a sparse set of points for a square undergoing an rigid transformation [11].



Frame 1 Frame 2 Optical flow 1-2

Figure 2.5: Changes in the grid pattern from frame 1 to 2 can be represented by a vector field, that vector field is the optical flow. The black square from frame 1 undergoes a translation and rotation between frames 1 and 2. Only the optical flow for the corner points (marked with grey circles) is shown.

One way to formulate the problem (as done by Lucas and Kanade) is to consider a small area around the image point (determined by ω_x and ω_y), and define the following function:

$$\epsilon(\mathbf{d},\mathbf{A}) = \sum_{x=-\omega_x}^{\omega_x} \sum_{y=-\omega_y}^{\omega_y} \left(F_n(\mathbf{x}+\mathbf{u}) - F_{n+1}(\mathbf{A}\mathbf{x}+\mathbf{d}+\mathbf{u})\right)^2 . \tag{2.4}$$

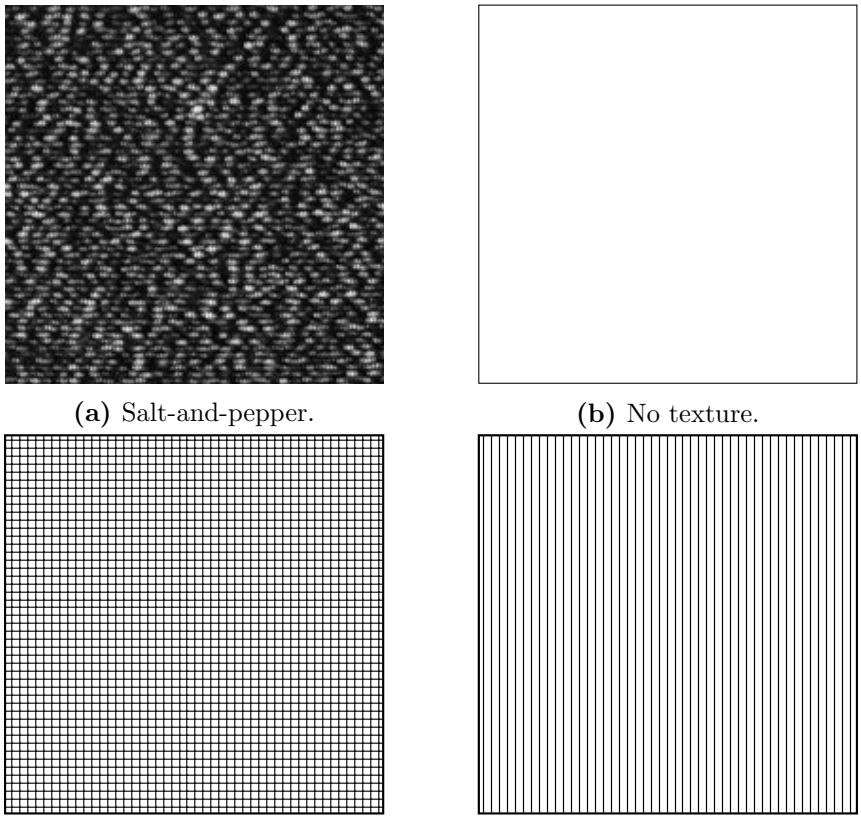
In this $\mathbf{A}$ is an affine transformation matrix that takes into account that the motion between the two images might not be a pure translation and $\mathbf{d} = [\delta_x,\delta_y]^T$. The goal is then to find $\mathbf{A}$ and $\mathbf{d}$ such that $\epsilon(\mathbf{d},\mathbf{A})$ is minimised [11, 19]. The optical flow (velocity) for the point $\mathbf{u}$ is thus $\mathbf{d}$. This works under the assumption that the optical flow around a given pixel is essentially constant.

This is a differential method for solving the optical flow and a pyramidal variant of the Lucas-Kanade method is implemented in OpenCV and is described in detail by Bouguet [11].

2.2.1 Optical flow tracker

To construct a tracker around the Lucas-Kanade method for optical flow a couple of additional components are necessary. Suitable feature points to track by calculation of the optical flow need to be located. To get good tracking one should also try to ensure that the points actually belong to the object which is to be tracked and not the background. That a point is easy to track does not mean that it is one that is of interest.

A good candidate method for detection of suitable features is the Shi-Tomasi corner detector as it is designed to select features based on how suitable they are for tracking [20]. Good features to track are in general regions which contain much motion information. Straight edges and unidirectional patterns can only be used to determine motions in one direction, corners and salt-and-pepper textures are more suitable as they are able to provide more information about the motion [21]. See figure 2.6 for examples.



(a) Salt-and-pepper. **(b)** No texture.

(c) Corners. **(d)** Edges.

Figure 2.6: Examples of different textures that can be on objects. Salt-and-pepper and corners are good for tracking. The other textures are hard to find the correct optical flow for and are thus bad for tracking.

The basic idea of corner detection is that given a small image patch, shifting the region slightly in any direction should give a large change in appearance. For uniform region there is almost no change regardless of which direction the patch is shifted, for straight edges there is only a large change if the shift is orthogonal to the direction of the edge. A mathematical approach to finding corners and edges is the Harris corner detector (which the Shi-Tomasi corner detector is a variation of) [22].

Making sure that features actually belong to the tracked object is harder. One option is to perform image segmentation to get rid of the background, this can be done through background subtraction or by simply finding contours and guessing which

contour belongs to the tracked object.

A simple tracker based on these ideas can then be formulated as follows:

1. Find the largest contour in a selected region of interest, this can be done by running Canny edge detection followed by a border following algorithm.
2. Locate good features to track inside the contour using the Shi-Tomasi algorithm for corner detection.
3. Calculate the optical flow for the corners to locate them in the next frame, if many points are lost try to redetect by going to 1, using last known tracking rectangle as region of interest.
4. Centre the tracking rectangle on the centre point of all tracked features.
5. Scale the tracked rectangle based on how much the points have moved in relation to the mean position of all points since last frame.
6. Goto 3.

This is the basic idea of the algorithm, in its simplicity it works quite well and is very fast.

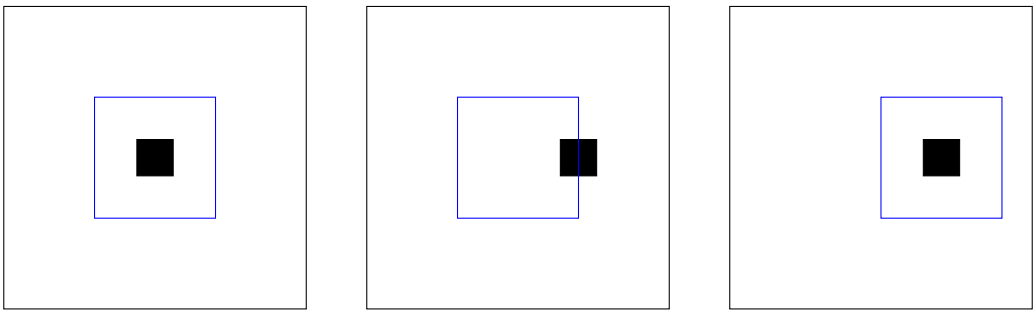
2.3 Mean shift

Another approach to tracking is by a technique commonly referred to as mean shift (MS). It is a mode seeking algorithm that works by mapping all pixels in an image into some feature space and then locate clusters. A cluster will then correspond to an image segment. The method is iterative and works by stepwise increasing similarity with a reference by shifting a window [23]. The mapping can be thought of as a greyscale image where high pixel values (darker) correspond to a higher likelihood that the pixel at that location belongs to the reference. The objective is then to centre the largest cluster, maximise the number of dark pixels in the window, this is illustrated in figure 2.7.

The mapping of pixels to a feature space can be done by histogram backprojection. The histogram for an initial area containing the object that is to be tracked is computed. Pixels are then assigned a probability of belonging to the object based on how many pixels of that colour exist in the reference [24]. If the histogram for the tracked object shows that the object for example has 66 % blue pixels, 33 % red pixels and 0 % green pixels. All blue and red pixels will be given a high probability of belonging to the tracked object, for an example of this mapping see figure 2.8. By mean shift the region with high probability pixels gets centred.

The equation describing the mean shift is

$$m(x) = \frac{\sum_{s\in\Omega} K(s-x)s}{\sum_{s\in\Omega} K(s-x)} \tag{2.5}$$



a) b) c)

Figure 2.7: Example of how mean shift works. Consider a) as the first frame and the black square in the centre is a cluster, b) is a frame where the cluster has moved. The largest cluster is at the far right of the tracking box and it is most likely that the object sought is there and thus the mean shift algorithm will stepwise move the box right until the cluster is centred again.

where $K(\cdot)$ is a given kernel function and Ω a neighbourhood to x. In the example in figure 2.7, x would be the centre of the rectangle, Ω all the pixels in the rectangle and the movement of the rectangle to the right be the update $m(x) \leftarrow x$. This is iterated until $m(x)$ converges.

2.3.1 ASMS: Robust mean shift with scale adaption

Adaptive scale mean shift (ASMS) is an improved mean shift tracker described in detail in [16]. It is one of the algorithms which this thesis evaluates for PTZ tracking and checks if it benefits from pre-processing in the form of background subtraction.

In mean shift tracking the kernel is moved from a given location $\mathbf{y}_0$ to a new location

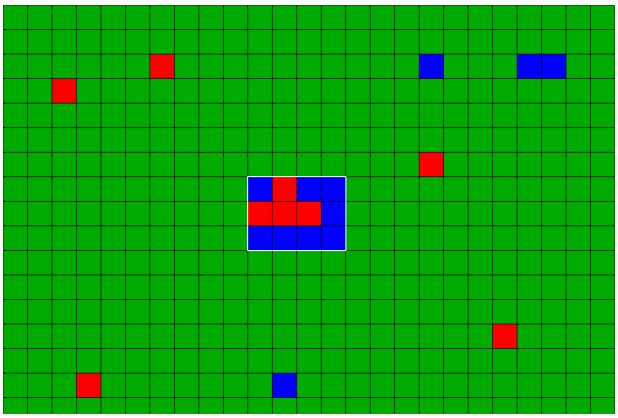
$$\mathbf{y}_1 = m(\mathbf{y}_0) = \frac{\sum_{i=1}^{n} w_i g\left(\left\|\frac{s_i - \mathbf{y}_0}{h}\right\|^2\right) s_i}{\sum_{i=1}^{n} w_i g\left(\left\|\frac{s_i - \mathbf{y}_0}{h}\right\|^2\right)} \tag{2.6}$$

Using the same notation as in equation (2.5) we have $s \in \Omega$ and n is the number of elements in Ω and where x was a coordinate in some space, here $\mathbf{y}$ is a point in two-dimensional Euclidean space. And $g(x) = -\frac{d}{dx}k(x)$ where $k(x)$ is some isotropic, convex and monotonically decreasing kernel profile.

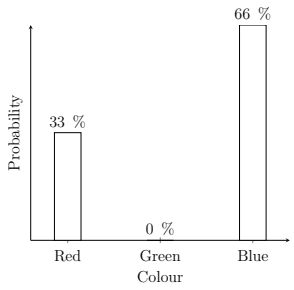
The weights

$$w_i = \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\mathbf{y}_0)}}$$

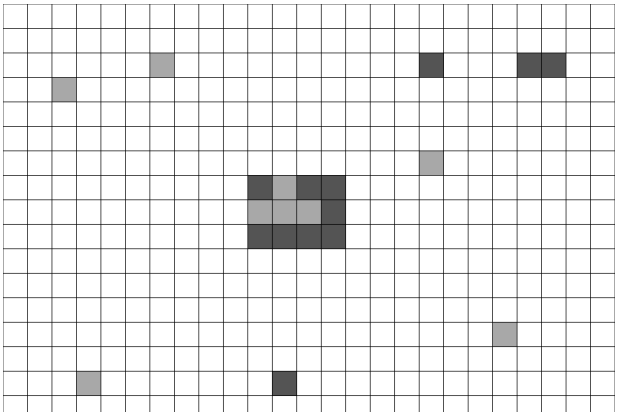
with u corresponding to the feature pixel s_i maps to represents the likelihood that s_i belongs to the target. $\hat{q}_u$ is the probability of feature u belonging to the target and $\hat{p}_u(\mathbf{y})$ is the probability of feature u in a target candidate centred at $\mathbf{y}$ [16]. This is improved



(a) Frame with selection rectangle.



(b) Colour histogram.



(c) Probability map.

Figure 2.8: Example of using histogram backprojection to convert a colour image to a probability map. In 2.8a an area is selected and a histogram describing the colour distribution is created. In 2.8c the probability mapping based on the histogram is shown. Darker pixels correspond to higher probability pixels.

in [16] by changing the calculation of w_i by taking an estimated background histogram into account through background ratio weighting (BRW). A guess for the background histogram is calculated from the neighbourhood of the tracked object.

The algorithm described in [16] is scale adaptive. A parameter describing the scale of the target rectangle is introduced and in much the same ways as for finding the mean shift a correct scaling is iteratively calculated. To prevent incorrect changes the calculated change from step $t-1$ to t is validated by calculating the change from t to $t-1$. If they do not give the same result the change is rejected and a weighted combination of the previous size, the estimated size and a "default size" is used instead.

The idea of the tracker is to take a simple mean shift tracker (as described in section 2.3) and enhance it by making it able to adapt to scale changes in a robust way. It is further improved by a method for histogram colour weighting that helps discriminate

the object from the background (BRW). The result of this is that features that have a high probability of both belonging to the background and the foreground are suppressed.

A simple example, your feature space is just three colours and you have an equal distribution of all the colours in the template target. If your background is dominated by one of these colours, BRW can compensate for this and downweight the background colours in the template target. For an illustration of this example see figure 2.9.

The complete algorithm for each frame is:

1. Calculate new position $\mathbf{y}_t$ and scale h_t.
2. Apply corrections to h_t.
3. If the change in scale is large than some threshold, perform consistency check (backward validation).
4. Goto 1.

2.4 Correlation filter

To find if a template is present in an image a correlation filter is used to correlate the template with the image, a response will indicate that the template is present. For example, if the template consists of an ellipse, the correlation between the filter and the image should produce a high response at the centres of ellipses in the image. A template is the part of an image which include the target object that is to be tracked. It is defined as the manually chosen object bounding box in the first frame, but can then be updated as combination of how the object looks like in subsequent frames.

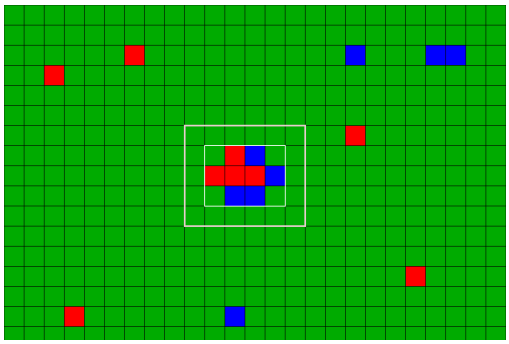
A tracker is constructed around this idea by performing tracking by detection. Samples are selected from an image and a classifier decides whether a sample is similar to the correct template or not (a sample can be any extracted part of the image). The classifier is trained to correctly classify negative and positive samples [17].

The training of the filter is based on regression

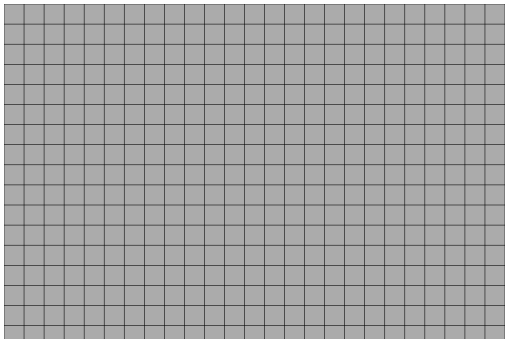
$$\min_{\mathbf{w}} \sum_{i} L(f(\mathbf{x}_i) - y_i) + \lambda ||\mathbf{w}||^2 \tag{2.7}$$

where L is a loss function, λ controls the amount of regularisation. $\mathbf{x}_i$ are the samples and y_i the regression target. The goal is to find the $\mathbf{w}$ that best describes the object. $f(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i$ is called linear regression and even further, using a quadratic loss function the problem is called ridge regression.

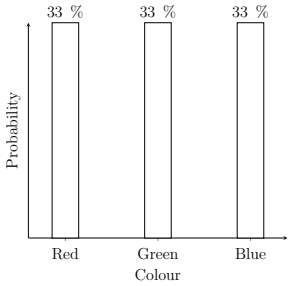
Samples to be evaluated in the next frame can be chosen differently, i.e. randomly or by a certain method. It is showed that if every possible sample from the next frame is chosen in the neighbourhood of the previous template position (dense sampling), circulant matrices can be used in the calculations. Two overlapping patches will contain some information that is the same in both due to the overlap [17]. A circulant matrix is diagonalised by the Discrete Fourier Transform (DFT). Diagonal matrices are good for fast calculations since several matrix operations are reduced to element-wise operations.



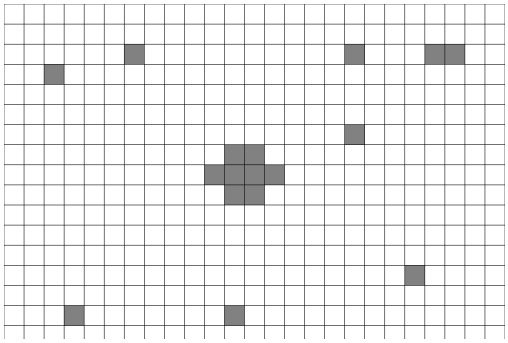
(a) Frame with selection rectangle. The enlarged outer rectangle is used when creating the background histogram.



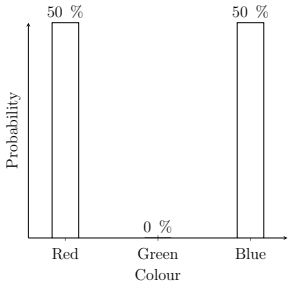
(b) Probability map.



(c) Colour histogram.



(d) Weighted probability map.



(e) Weighted colour histogram.

Figure 2.9: Example of "background ratio weighting", an area around the template is selected to get an approximation of the background colour distribution. This information can then be used to weight the template histogram.

The Fourier transform is periodic but to reduce the effects of a wrapped-around image boundary, cosine windowing and zero-padding is used. In a $n \times n$-image a cosine

window downweights values such as $x \in [0,1]$ closer to the border according to [25]

$$x_{ij} = (x_{ij}^{raw} - 0.5) \sin\left(\frac{\pi i}{n}\right) \sin\left(\frac{\pi j}{n}\right) \qquad \forall i,j = 0, \ldots, n-1. \tag{2.8}$$

A Histogram of Oriented Gradients (HOG) is a way to represent an image area, such as a 2×2-pixel square area. The main idea is to calculate the image gradient in each pixel and then building a histogram of these gradients, see figure 2.10. Doing this can give better features to use and the cost of computing them is countered by reducing the amount of features the tracker operates on. Using a cell size of 2×2-pixels for the HOG reduces the amount of features by a fourth compared with using the raw pixel values.

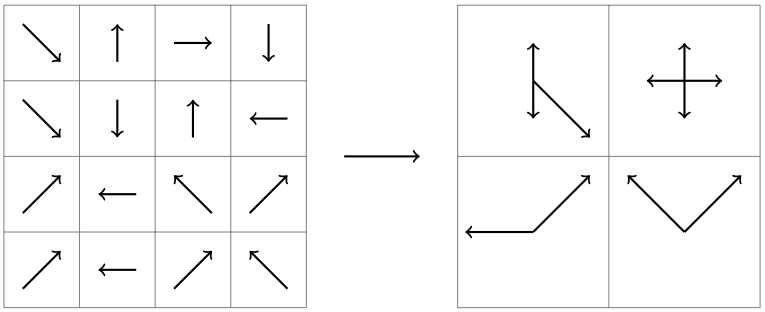


Figure 2.10: To the left, image gradients. To the right, Histograms of Oriented Gradients (HOGs) for each corresponding area of 4 pixels.

When using more flexible and powerful non-linear regression functions $f(\mathbf{x}_i)$ the kernel trick can be used to make the calculations cheaper. The problem is solved like it is linear but the variables are different, so called dual variables. The kernel trick is that one never computes a data vector in the high-dimensional space described by the kernel function, but always operate on inner products between data vectors in the high-dimensional space [26].

2.4.1 KCF: Kernelized correlation filter

KCF is an tracking-by-detection algorithm described in detail in [17]. It operates in the Fourier domain, and use non-linear regression with a Gaussian kernel with the help of the kernel trick. The dense sampling procedure makes it possible to calculate the responses for all samples at the same time.

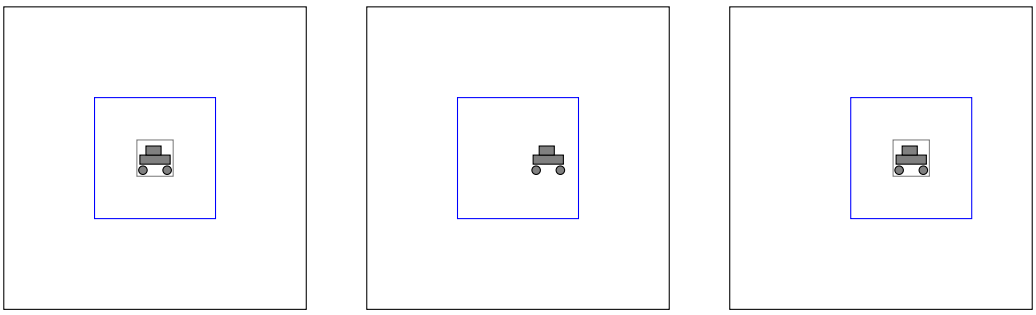
The algorithm is:

1. In the current frame, within the image region that was the previous template position, samples are all possible shifts of a window of the same size as the template, with wrap-around at the edges. Cosine-windowing and zero-padding is used.

2. The exact target position is calculated as the sample with calculated maximum response in this window.

3. An updated model of the target is trained on the current frame and the classifier is then updated.

KCF can be used on raw pixel values as well as multi-channel features such as Histogram of Oriented Gradients (HOG), see figure 2.10. HOG features seem to improve the performance of KCF relative to operating on raw pixel values [17].



Frame 1 Frame 2 Frame 2 updated

Figure 2.11: An initial target is selected in frame 1, an enlarged rectangle around it is used as template patch. In frame 2 the blue rectangle is the starting guess for finding the object, the regression function is evaluated for all cyclic shifts of that patch to find the one most similar to the training patch. The position of the patch and target is then updated.

2.5 Measures of tracking performance

Two dimensions of tracking performance can be expressed as accuracy and robustness. Accuracy is how close the tracker is, when it is tracking. Robustness is how good the tracker is at not losing the track. The region overlap is an accuracy measure defined as the overlap between the ground-truth region and the region proposed by the tracker. The region overlap captures the tracking accuracy in terms of the position as well as the size of the tracker region. The failure rate is a robustness measure defined as the number of frames where the tracking is lost and then reinitialised (the tracker box is set as the ground-truth). Frames must be annotated and especially the accuracy measure is therefore influenced to the subjective opinions of the annotators, especially for objects that do not have a well-defined boundary or centre. Then the frames must be annotated several times by different annotators and averaged over [27].

Visual tracking is done in different scenarios and in different ways. The tracking that the VOT Challenge [8] aims to analyse is in the field of single-camera, single-object, model-free and causal trackers. Model-free tracker means the tracker only has the bounding box in the first frame to learn the object to be tracked. A causal tracker does not use future frames to track the object in the current frame.

To analyse tracker performance, an annotated dataset is beneficial. Annotation is done by labelling each frame with attributes, such as illumination change, camera move-

ment and size change. Then it is easy to make sure that the sequences in the dataset have a large variety of attributes, assuring a fair comparison of general tracking performance. In the VOT Challenge the tracker is reinitialised when tracking has failed, tracking redetection is thus not evaluated. Trackers are run several times on the dataset to get relevant measures, since trackers can be stochastic. The tracking performance is measured in robustness and accuracy. The accuracy during the first frames following after the reinitialisation could be better than expected due to bias, and therefore the ten first frames are not included in the calculation of the accuracy result. After tracking failure the tracker is reinitialised five frames later, since the tracker might fail instantly due to the same attribute, such as occlusion, and this should not be reflected in the robustness measure [28].

3

Method

This chapter serves to outline the work done during the thesis and describe how the results were obtained. The first step of the work was to get acquainted with computer vision as a field of research and the state-of-the-art in motion tracking. This was done by a literature review, especially [6, 8, 28, 29] was used to find suitable candidates for evaluation. From this two state-of-the-art trackers were chosen (ASMS [16] and KCF [17], theoretical background described in 2.3.1 and 2.4.1) that seemed both fast and quite robust. In addition to these, one simple tracker based on keypoint tracking through calculation of optical flow was implemented (see section 2.2.1).

A system for performing the testing of algorithms was developed, written in C++ and making use of the library OpenCV for the image processing. The implementations of ASMS and KCF are slight modifications of publicly available code[1,2]. The code was modified to give a consistent interface for all the tracking algorithms and to make it possible to use them together with background subtraction.

For initial testing the trackers were evaluated in the VOT Challenge framework on the dataset from the 2015 competition. The results from the competition are available so it is possible to compare the results of the implementations from this thesis with that of the original algorithm authors.

In addition to the framework for the tracking algorithms, software for controlling the pan-tilt unit was developed to make it possible to control the camera and record video sequences annotated with absolute pan, tilt and zoom positions. The PTZ camera and robot can be controlled either manually by mouse or keyboard, or by one of the tracking algorithms. The control is based on a feedback loop for the pan-tilt motions in order to keep the tracked object centred in the image.

An algorithm for background subtraction was implemented based on an article by Elgammal et al. [15]. One implementation of the original algorithm is available as a

[1] `https://github.com/vojirt/asms`
[2] `https://github.com/joaofaro/KCFcpp`

part of the BGSLibrary[3], the implementation used in the testing in this thesis is entirely based on the written article and modified to work with a moving camera. Not all features described in [15] were implemented, for a description of the algorithm used see 2.1.

The trackers were evaluated to determine whether they benefit from background subtraction. Details of the testing procedure are described in 3.2.

3.1 Materials

The hardware the tracker system consists of is a camera with zoom, a pan-tilt unit and a computer for control. A schematic image of the camera system is shown in figure 3.1. The system has two axes of rotation, one for pan and one for tilt. Note that the pan and tilt axes do not go through the entrance pupil of the camera but are a bit offset. This means that a pure rotation of the camera can not be done, the camera will translate as well when rotating.

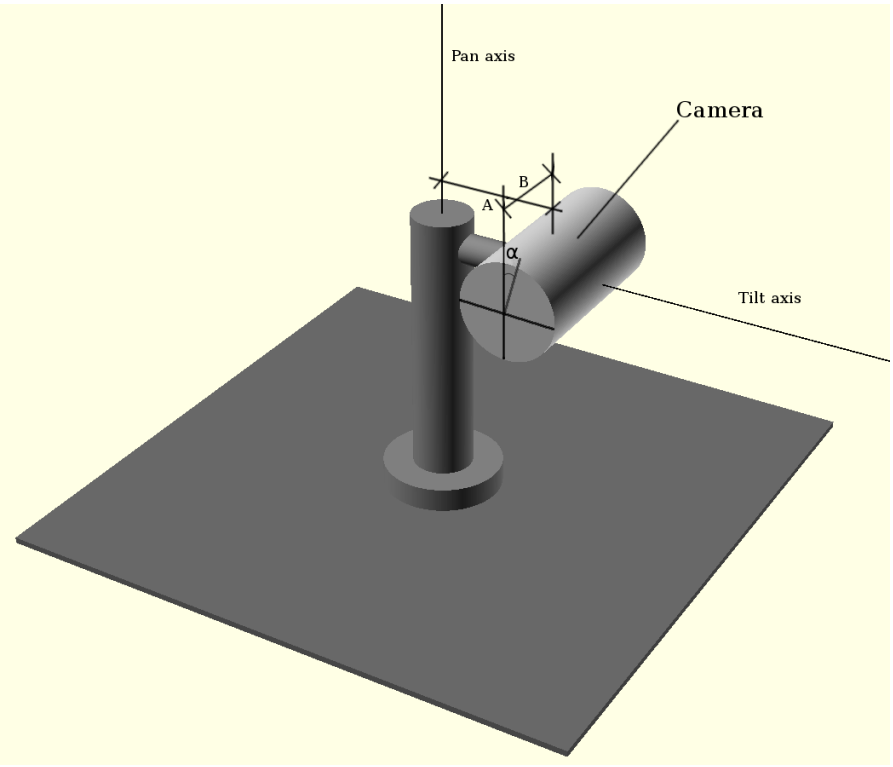


Figure 3.1: Illustration of the camera and pan-tilt unit used. A and B are offsets from the rotational axes and α is the rotation of the image sensor compared to the pan-tilt coordinate system, exaggerated in this image.

[3] `https://github.com/andrewssobral/bgslibrary`

3.2 Testing

The purpose of the testing is to determine the impact of background subtraction on tracking and to evaluate how well it is possible to perform background subtraction given a sequence where the camera rotates. A dataset was created for the evaluation by manually controlling the pan-tilt unit and recording multiple sequences with per frame annotation of pan-tilt position.

To get a baseline tracking performance each tracker was evaluated on the VOT2015 Challenge[4] without using background subtraction (no prior knowledge about camera movement available). The main reason to do this was to check that the implementations of the algorithms we used performed as expected and to make it possible to compare OPTFLOW with the others.

3.2.1 Background subtraction

The resulting background mask from the background subtraction is used in different ways for the trackers. For ASMS and KCF a new image is created from the original image by setting background pixels to black. This image is then input to the tracker. For OPTFLOW, feature points on the background were removed before tracking.

The performance of the background subtraction is evaluated on different cases to see what impacts its performance. Two cases are constructed. The first is a simple sequence, taking a single large image and creating a video by sweeping over it with a smaller window. In this case all pixels should be classified as background since there are no moving objects in a static image.

The second case is to evaluate the result of the subtraction when there is no error in the data for the camera movement. This was done by recording a sequence without moving the camera and then constructing a new video using small parts of the original sequence (moving a window over it, simulating a moving camera). By doing this we minimise vibrations and we get perfect knowledge of the per frame movement.

Finally, the background subtraction is evaluated on sequences from the PTZ camera under the conditions: only pan motions, only tilt motions, and both pan and tilt motions. For this background model, small angle rotations are assumed and the camera movement is approximated as a translation.

3.2.2 Camera parameters

To perform background subtraction when the source of the frames is moving, knowledge about the pixelwise offset between the frames is needed. One way to get that without any prior knowledge about the movement is by tracking points belonging to the background from one frame to the next, by for example optical flow.

When knowledge about the camera movement is available, some way to relate changes in pan and tilt angle to changes in pixel position in an image is needed. When the

[4] `http://votchallenge.net`

changes in pan and tilt are small, the change in pixel position can be approximated to be proportional to the change in pan and tilt.

$$\Delta x = c_1 \cdot \Delta p$$
$$\Delta y = c_2 \cdot \Delta t$$

The coefficients c_1 and c_2 can be estimated by for example using optical flow to get an estimate for the pixel movement and compare that with the change in pan and tilt. They can also be found by manually matching images with known camera position and calculate the coefficients from that. Both methods are evaluated.

3.2.3 Test procedures for evaluating the trackers with background subtraction

- **Scenario 1: Tracking with background subtraction.**
  A sequence of frames is collected from a static camera. In the sequence there is at least one moving object. A moving camera is simulated by constructing a new sequence of frames where each frame is a fixed size region from the corresponding static camera frame. The pixel position of the extracted region is logged in order to simulate translation information from the robot. Then for each tracker:
  - Initiate a bounding box on the object to be tracked.
  - Track with and without background subtraction.
  - If the tracking is lost, reinitialise by giving a new bounding box around the object.
  - Count the number of times the tracking is lost and how many frames processed per second.

- **Scenario 2: Simple tracking with background subtraction and robot.**
  Sequences of frames are collected from the PTZ camera when it is moved manually, in the sequence there is a moving unicoloured circle in front of a simple background. Pan and tilt positions are logged to make it possible to align the frames. Sequences:
  - a. The camera is only panned right and left.
  - b. The camera is only tilted up and down.
  - c. The camera is panned and tilted in an irregular pattern.
  - d. The moving object is moved around in an irregular pattern, the camera is manually controlled to keep the object centred in the image (as the tracker would control it).

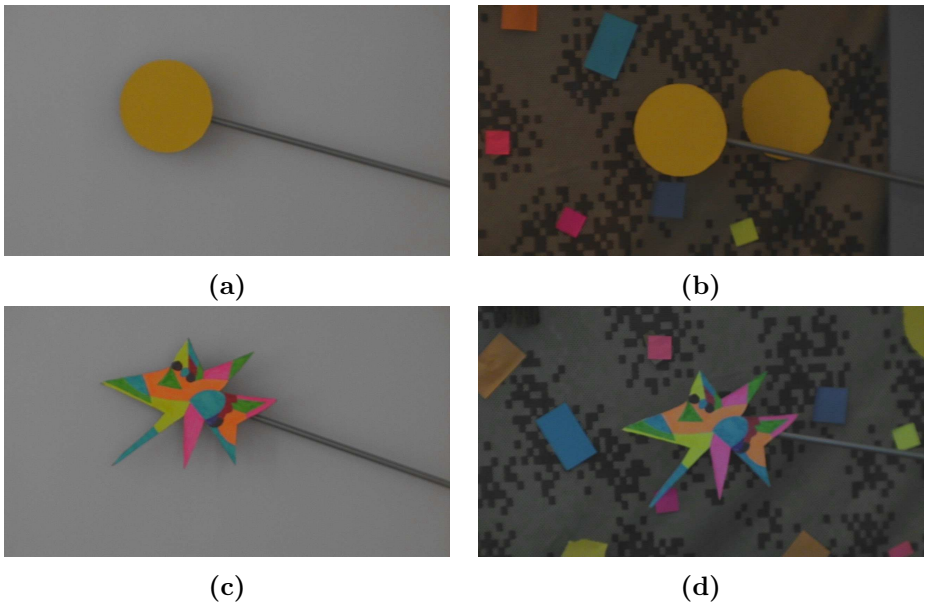
  For each tracker:

- Initiate a bounding box on the object to be tracked.
- Track with and without background subtraction.
- If the tracking is lost, reinitialise by giving a new bounding box around the object.
- Count the number of times the tracking is lost and how many frames processed per second.

- **Scenario 3: Tracking with background subtraction and robot**
  - Same procedure as scenario 2 but with a more complicated background with clutter.

As targets for the tracking, a yellow circle and a multicoloured polygon were selected. The circle was chosen to test with a simple shape, invariant to rotation, and a single distinct colour. The polygon was chosen to get an object as different as possible to the circle, with several colours and many corners.

The initial bounding box were chosen as the smallest upright rectangle so that every part of the object was inside the box. Sometimes the entire object may move outside the camera view, this is not reported as a failure. The bounding box is initialised around the object when it enters the view again. Example images from all the sequences can be seen in figure 3.2.



(a) **(b)**

(c) **(d)**

Figure 3.2: All the different combinations of moving test objects and backgrounds used for evaluating the trackers with background subtraction.

4

Results

In this chapter, system and test data are presented. An in-depth discussion of the results follows in the next chapter.

4.1 Camera configuration

The camera is configured to have a resolution of 1280×720 pixels and that a step in pan or tilt position correspond to a change in angle of $\frac{1}{30}^\circ$.

By recording a sequence of frames and storing the absolute pan and tilt position for each frame the relation between change in pan position and pixel movement in the image was calculated. Another sequence was also recorded where in addition to the absolute pan-tilt position for each frame an estimate of the change in pixel position between successive frames was stored. The estimate was calculated by keeping track of the optical flow for some features and taking the average change in position as an estimate for the entire frame movement.

For both sequences the optical zoom was set to $\times 10$. Manually matching frames the following correlation was found:

$$\Delta x = 5.48 \cdot \Delta p \tag{4.1}$$
$$\Delta y = 5.63 \cdot \Delta t \tag{4.2}$$

where Δp is a change in pan position and Δt is a change in tilt position.

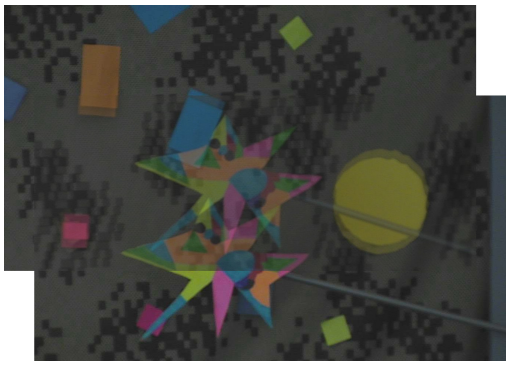
From the sequence with the recorded estimates of Δx and Δy the following correlation was found by fitting a line to the data minimising the mean squared error:

$$\Delta x = 5.39 \cdot \Delta p \tag{4.3}$$
$$\Delta y = 5.34 \cdot \Delta t. \tag{4.4}$$

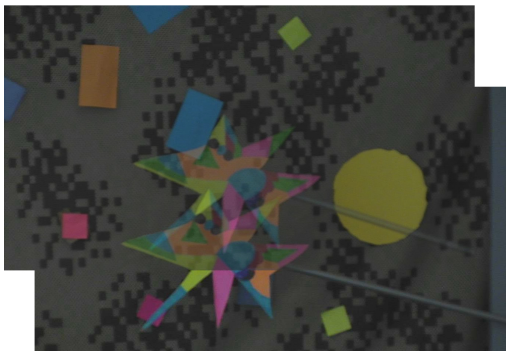
In the testing the coefficients 5.48 and 5.63 were used. The other coefficients were also tested but there was no distinguishable difference in the subtraction result.

4.2 Background subtraction

To use the images from a moving camera to build a background model the images must be positioned so that a pixel that has not moved is at the same place. An example of how well this works for the camera used can be seen in figure 4.1a together with how it would have looked if the match was perfect. Constructing a video sequence from one single large



(a)



(b)

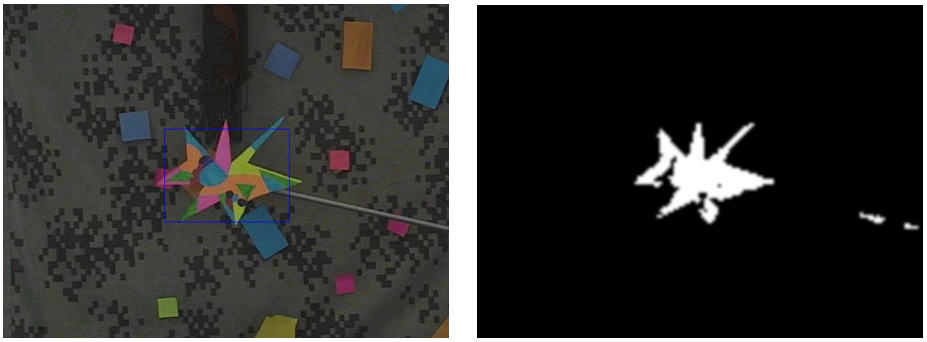
Figure 4.1: The result of using the change in camera position (a) to line up two images with 30 frames between and manually matching them (b). The angular distance between the frames is $(\Delta p, \Delta t) = (0.45°, 1.38°)$

image gave the expected result, all pixels were classified as background. By calculating the absolute difference between frames and setting the threshold to zero we noticed that some errors were introduced in the sequence by the compression of the images when the sequence was constructed. Pixel intensity values could differ on approximately 1 % when compared to the source image.

4.2.1 Stationary image

The background subtraction was evaluated on a sequence with simulated camera movement, the camera was stationary and the movement was simulated by constructing a new sequence from parts of the entire frames. This minimised errors in the background subtraction due to possibly incorrect alignment of frames, since the per frame offset was known pure translations.

Even if the alignment of the frames is perfect the background subtraction is not. In figure 4.3 there are two examples of how it can go wrong. The object can accidentally become a part of the background model, if the object then returns to the same position it erroneously is classified as background. Another problem is that the background subtraction algorithm only operates on a colour level, if a moving object moves over a background that has the same colour the overlapping part will be classified as background. There is a tradeoff between making the subtraction robust against slight variations in colour (for example a change in lighting) and making correct classifications when there are similar objects with only a slight difference in colour.



(a) **(b)**

Figure 4.2: Original image (a) and the background mask (b) from a sequence where the camera movement is simulated.

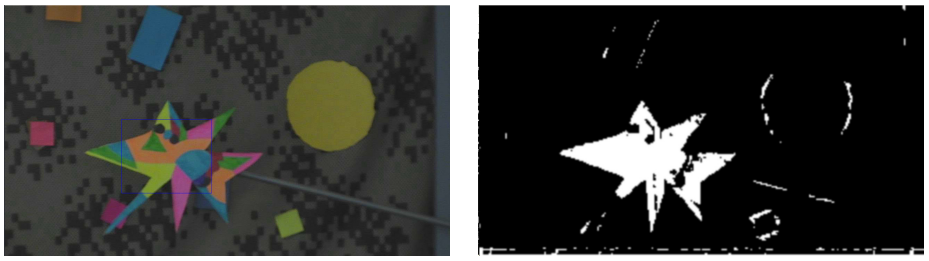
4.2.2 Rotating camera

For the moving camera the same problems with background subtraction as with the stationary camera exist. In addition there are errors due to incorrect alignment of frames. This increases the amount of pixels classified as foreground due to the difficulty in separating movement from vibration and actual robot movements. Another problem is that details get smoothed out in the background model. A patch that is quite small if you only look at a single frame can appear to cover a much larger area, this is especially troublesome if the patch has the same colour as the moving object. Figure 4.5 demonstrates this, a yellow circle moves over a background that contains a small yellow rectangle.

The subtraction was evaluated for the camera moving freely as well as restricted to only pan and only tilt but no visible difference was noticed for the different cases.

(a) (b)

Figure 4.3: Problems with the background subtraction that occur even though the frames have been aligned perfectly. In 4.3a the object have become a part of the background model and in 4.3b the object passes over a patch of background with the same colour. The object is a yellow circle.



(a) (b)

Figure 4.4: Frame 250 and the corresponding background mask from the sequence star10. Some edges from background objects are classified as foreground due to missalignment.

Figure 4.5: Behind the yellow circle is a small yellow square that has become smoothed out and appears much larger in the background model due to errors in the alignment of frames. The circle is in the same position as in figure 4.3b for comparison.

4.2.3 Speed

The impact of different parameters for the background subtractor was evaluated by running the same sequence multiple times with different settings while monitoring the frame rate. All parameters except W (how often frames are added to the background

model) impacted how fast the subtraction was. The size of the image also heavily impacts the performance. This is as expected, the number of pixels directly correspond to the amount of computations and W is the only parameter that does not influence this. Using the same settings as in column 1 in table 4.1 but using an image sequence at a lower resolution (160×120 instead of 320×180) the frame rate was 130 fps, compared to 56 fps.

Table 4.1: Average frame rate
Impact of different parameter settings on frame rate, evaluated on a sequence with 1280×720 resolution downscaled to 320×180. N_{st} and N_{lt} are parameters for the number of frames in the short and long term models, W governs how often the long term model is updated, n_{pyr} sets the number of levels in the image pyramid and th is a threshold a pixel probability must exceed to be classified as background.

N_{st}	15	15	15	2	25	20	20	20
N_{lt}	5	5	5	0	0	5	5	5
W	60	30	60	–	–	30	30	30
n_{pyr}	2	2	1	2	2	2	2	2
th	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-4}	10^{-6}	10^{-8}
Time per frame [ms]	17.9	17.9	22.7	12.2	21.3	31.3	20.4	19.2

4.3 Tracking

The trackers were evaluated on four different object-background combinations. A multicoloured polygon on a white background and on a cluttered background, and a unicoloured circle on the same two backgrounds. The results in number of failures is presented in table 4.2.

Multiple sequences were recorded for the different combinations, in total 21 sequences of varying length were produced. The main difference between the recordings was how the PTZ was allowed to move.

The backgrounds were chosen so that one was very simple with large contrast against the tracking target without features that could appear as moving due to an incorrect alignment of frames in the subtractor. The other was selected to be the opposite, containing a lot of small and large features and areas with the same colour as the tracking target. The objects were chosen in the same manner, one that was simple with uniform colour and one with some texture.

4.3.1 Speed

The average time it took to process a frame for the different trackers for the different sequences is shown in table 4.3. KCF and OPTFLOW showed quite stable performance over all the sequences while ASMS had a drop in frame rate for the sequences with the

Table 4.2: Tracking without background subtraction. Stars indicate sequences with cluttered background.

Sequence	**OPTFLOW** Failures	Time per frame [ms]	**ASMS** Failures	Time per frame [ms]	**KCF** Failures	Time per frame [ms]	#frames	Size
circle1	0	3.1	0	8.2	0	21.3	1309	640×480
circle2*	6	3.5	1	10.0	0	21.7	1860	640×480
circle3	0	6.2	0	33.3	0	20.8	1227	1280×720
circle4*	2	6.8	0	25.0	0	14.9	2774	1280×720
circle5	0	5.3	0	25.0	0	20.4	1658	1280×720
circle6*	1	6.3	0	34.5	0	19.2	3443	1280×720
circle7	0	5.6	0	25.6	0	21.7	918	1280×720
circle8*	3	6.5	0	31.3	0	20.0	2727	1280×720
circle9	0	5.5	0	23.3	0	22.2	1437	1280×720
circle10*	1	6.5	0	34.5	0	23.3	2554	1280×720
circle11*	7	6.6	1	38.5	0	18.9	3784	1280×720
star1	0	3.4	0	14.7	0	17.9	971	640×480
star2*	0	3.2	0	14.5	0	18.2	1757	640×480
star3	0	3.6	0	83.3	0	23.3	1325	1280×720
star4*	0	3.6	0	142.9	0	20.0	1753	1280×720
star5	0	6.5	0	62.5	0	18.2	2064	1280×720
star6*	0	6.2	0	111.1	0	15.2	2348	1280×720
star7	0	6.5	0	55.6	0	15.9	1596	1280×720
star8*	0	6.2	0	62.5	0	16.7	3999	1280×720
star9	0	6.1	0	55.6	0	16.7	1446	1280×720
star10*	0	5.9	0	142.8	0	16.1	3277	1280×720
total	20		2		0		44227	

star-shaped object. Looking at the frame rates reported in table 4.5 for the sequences with lower resolution it seems that both ASMS and OPTFLOW are faster on smaller images. KCF on the other hand reports a stable performance regardless of resolution size.

Table 4.3: Average time per frame [ms]

Combination	OPTFLOW	ASMS	KCF
circle on white	5.7	26.8	21.3
circle on clutter	6.5	32.8	19.3
star on white	5.7	64.3	18.5
star on clutter	5.5	114.8	17.0

4.3.2 VOT Challenge

The trackers were evaluated on the VOT2015 dataset. Table 4.4 shows the results with respect to robustness. Robustness is measured by counting the number of times a tracker fails on a sequence and then taking the average over multiple runs.

Only looking at robustness all trackers performed well on the sequences: bag, ball1, birds2, blanket, bmx, car2, dinosaur, fernando, fish3, godfather, iceskater1, iceskater2, marching, racing, sheep, singer1, singer3, sphere. These sequences had in common that the objects where quite constant in size, without abrupt changes in colour or form and the sequences contained little occlusion.

Looking at the sequences where the trackers performed badly, almost all trackers had problem with occlusion that lasted longer than one or two frames. KCF had trouble when objects got blurred due to rapid motion. This did not affect ASMS as much, as the colour content in an area is less distorted than the actual form during blurring. ASMS had trouble when there where multiple objects with the same colour or the entire frame had roughly the same colour (see soccer1 for example, red confetti obscuring a soccer team with red clothes). All trackers had problems when the contrast between background and the tracked object was low.

4.3.3 Tracking with background subtraction

The number of failures for tracking on a masked background is presented in table 4.5. OPTFLOW had 91 failures, compared to 20 without background subtraction. ASMS had only one failure in total, one less than tracking without background subtraction. KCF had 13 failures, without background subtraction there were no failures.

The background subtraction reports a speed of about 50 fps (20 ms), calculating the frame rate when combining a tracker and background subtraction is straightforward. The two stages of background subtraction and tracking are run sequentially and not in parallel so the time per frame can be added.

Table 4.4: Robustness
The table gives the robustness score achieved on each sequence in the VOT2015 dataset. For comparison, stationary is benchmark tracker that does not update the position of the target after initialisation.

Sequence	OPTFLOW	ASMS	KCF	Stationary	Length [#frames]
bag	0	1	0	3	196
ball1	1	0	0	9	104
ball2	5	2	3	4	40
basketball	2	1	1	12	725
birds1	8.2	3	2	18	339
birds2	0	2	0	3	539
blanket	1	0	0	1	225
bmx	0	0	1	1	76
bolt1	11.8	1	0	8	350
bolt2	2	1	1	9	293
book	5	4	9	11	175
butterfly	3.33	0	2	5	151
car1	3.07	1	0	12	742
car2	0	0	0	3	393
crossing	3	1	1	5	131
dinosaur	0	0	2	6	326
fernando	1	0	2	3	292
fish1	4	2	4	7	366
fish2	4	3	5	5	310
fish3	2	1	0	6	519
fish4	1	1	3	8	682
girl	4	1	1	14	1500
glove	4	2	3	9	120
godfather	0	2	0	3	366
graduate	6.13	4	4	12	844
gymnastics1	3	1	7	4	567
gymnastics2	2.07	0	5	4	240
gymnastics3	2	3	4	3	118
gymnastics4	1	2	2	3	465
hand	6	5	8	15	267
handball1	10	3	7	20	377
handball2	12	3	11	28	402
helicopter	0	2	1	2	708
iceskater1	0.33	0	1	5	661
iceskater2	0	0	3	2	707
leaves	6	0	5	7	63
marching	1.2	1	0	8	201
matrix	8	2	4	4	100
motocross1	1	2	2	4	164
motocross2	1	0	3	0	61
nature	2	3	3	6	999
octopus	1	1	0	1	291
pedestrian1	4	3	10	14	140
pedestrian2	4	1	1	15	713
rabbit	6	4	5	8	158
racing	0	1	0	3	156
road	11.3	4	0	20	558
shaking	1	1	1	2	365
sheep	0	3	0	4	251
singer1	0	1	0	4	351
singer2	2.53	1	1	2	366
singer3	1	1	1	5	131
soccer1	0	10	2	5	392
soccer2	15	3	14	16	129
soldier	1	2	1	2	138
sphere	0	0	0	6	201
tiger	2	2	0	20	365
traffic	1	2	0	0	191
tunnel	3.73	5	0	0	312
wiper	6.07	2	0	5	341

Table 4.5: Tracking with background subtraction. Stars indicate sequences with cluttered background.

	OPTFLOW	ASMS	KCF		
Sequence	Failures	Failures	Failures	#frames	Size
circle1	4	0	0	1309	640×480
circle2*	13	0	1	1860	640×480
circle3	13	0	2	1227	1280×720
circle4*	4	1	0	2774	1280×720
circle5	4	0	1	1658	1280×720
circle6*	3	0	2	3443	1280×720
circle7	2	0	1	918	1280×720
circle8*	5	0	1	2727	1280×720
circle9	3	0	0	1437	1280×720
circle10*	4	0	0	2554	1280×720
circle11*	7	0	1	3784	1280×720
star1	1	0	1	971	640×480
star2*	5	0	0	1757	640×480
star3	2	0	1	1325	1280×720
star4*	3	0	0	1753	1280×720
star5	5	0	0	2064	1280×720
star6*	1	0	0	2348	1280×720
star7	3	0	1	1596	1280×720
star8*	4	0	0	3999	1280×720
star9	4	0	1	1446	1280×720
star10*	1	0	0	3277	1280×720
total	91	1	13	44227	

5

Discussion

Using background subtraction should make some algorithms perform better in circumstances where they otherwise have trouble. For OPTFLOW it can be used as a tool to reject background features and for colour based trackers like ASMS, filtering out the background can increase the contrast. Correct background subtraction can be hard to perform, and if done incorrectly it can do more damage than help.

A better use of the information about the camera motion might be to use it to create a motion model for the tracked object. This would help the tracking by making it possible to give a better starting guess for locating the object in a new frame.

For the evaluation of tracking with a PTZ camera, test data are needed. To make tests fair and avoid overfitting to certain scenarios data need to be diverse and cover a wide range of scenarios. For general tracking there are a lot of sequences available for testing but data of the camera movement in the sequence are often not available.

Robustness is the primary factor of interest for the PTZ tracking system. As the camera moves, the only thing that really matters is that the tracked object stays in view. Accuracy is in a sense related to robustness. If the accuracy is good the tracker should have a better chance to keep it in view and thus be robust.

5.1 Camera

There are some problems with projecting a 3D scene onto a 2D plane and then stitch planes from different camera directions. The error from approximating the camera rotation as a translation is greater at low zoom levels as the angle of view for the camera is larger than for higher zoom levels.

While increasing zoom reduces some problems it amplifies other, e.g. vibrations from the servos controlling the movement become more visible. When using a real camera it is hard to make the image sensor perfectly aligned with the rotational axes. Depending on how the camera and the control unit is constructed the offset for the rotational axes

from the entrance pupil can be quite large. The camera used is illustrated in figure 3.1 and has offsets of varying size for all axes. What this results in is that rotations are also translations and that a change in for example pan position can give an offset in both x and y coordinates for the image. If the offset angle for the image sensor α is zero, a change in pan position should only give a translation along the x axis in the image.

The vibrations proved to be the factor that had the largest impact on the possibility to align frames correctly. Even if the factor correlating a change in pan and tilt position to a change in x/y position of a pixel was exactly known, vibrations introduce an error of a couple of pixels. This error had largest influence when moving very slow (the object moving a couple of pixels per frame) or when changing rotation direction.

5.2 Background subtraction

In theory background subtraction could be a good tool for pre-processing a video stream to get an input frame that is easier to track in. In practice performing background subtraction well is a non-trivial task even if the camera is static. Adding inaccuracies from a moving camera it is possible to do more harm than good. A hypothesis is that background subtraction would help a tracker in sequences where the tracked object passes close to a similar object that is not moving. Then the tracker may start to track the close-by similar object but background subtraction will suppress the static object. It seems that it is a rare case that all these requirements are fulfilled.

Looking at the literature concerning state-of-the-art background subtraction less work appear to have been done on subtraction with a non-static camera. Even in the static case there does not really exist any method that works well in all scenarios, adding the requirement for real-time speed reduces the potential methods even more [30, 31].

Scaling down the input images before performing the background subtraction is one way to reduce the computational load at the cost of losing details. A pyramidal implementation performs background subtraction first on a very coarse level and use that as a mask for performing the background subtraction on a higher resolution image. Such an implementation can reduce the number of pixels that need to be evaluated without losing too many details.

When used as pre-processing for tracking, the loss of details that occurs when reducing the scale can be beneficial as it increases the amount of false foreground (classifying background as foreground) along the edge of objects. For the tracking, false background are more harmful than false foreground so a trade-off where the amount of false foreground increases at the cost of less true background is good.

Another problem arises when the tracked object slows down and is about to stand still, then it will become part of the background model if not handled. By keeping track of the speed of the object and stop updating the background model until the object resumes its movement this can be avoided.

We noticed that the algorithm does not work well when a moving object passes over background that has the same colour. This is a fundamental problem in tracking and in this algorithm as it is only designed to care about changes in colour for a pixel. Only

detecting the edge of moving objects that have large areas of one colour can be solved by making sure that the object itself is not incorporated in the background and that the blind update model has large enough amount of skipped frames between updates. No such thing can be done to solve the problem that the background has the same colour as the moving object. A way to handle it is to notify the user that the object is entering a background region that has the same colour as the object.

The background subtraction seems good when exploring new areas in images. When it returns to a place it has been stationary at for a while earlier in the sequence, it gets reported as background. This is because the object has been saved as background in the blindly updated history. Having the camera restricted to tracking a object in a small area where it revisits the same spot multiple times might be unfair as that is a scenario that is hard to handle well and maybe not so common.

The way we used the background subtraction mask was that we set the background pixels to black and left the foreground pixels unaffected. Then we input this new image to the tracker. This means that the trackbox will consist of black pixels that get incorporated into the description of the tracked object. For example, there will probably be a sharp edge between the foreground and the background instead of a smoother transition. A large gradient will be introduced into the object description affecting the KCF tracker. ASMS constructs its template in the first frame before background subtraction, and background ratio weighting should be able to suppress the black pixels incorporated by the background subtraction.

5.3 Tracking

The recorded test sequences were easy for all trackers except the circular target sequences, which were hard for OPTFLOW. The reason OPTFLOW did not perform well on those sequences is that the target had no good features, e.g corners. The sequences had no occlusion or change in light and the target did not change appearance through the sequence as a consequence of out of plane rotation or movement towards the camera.

As all trackers performed acceptably on the sequences without using background subtraction the only thing we could show was whether the background subtraction would make the tracking worse. Running the trackers on the sequences when using background subtraction made the trackers, especially OPTFLOW, more prone to lose the target. This might be due to that the constructed scenarios are not sequences where background subtraction is suitable for helping the trackers, or due to how the information from the background subtraction is used by the trackers.

Take ASMS for example, it operates in the same feature space as the background subtraction algorithm (pixel colour values). The scenario where it might be reasonable to expect the background subtraction to help is when the tracking target moves close to background features that are at rest and that have similar colour distribution. Looking at how ASMS works this is not a scenario where the tracking is prone to fail.

OPTFLOW on the other hand might have more to gain from using background subtraction as it discards all colour information. Running the tests showed that it is very

sensitive to false background in the background subtraction. The way the information from the background mask is used is to look at the position for each tracked feature in the background mask, and if that position is background, discard the feature. This is good for getting rid of points that belong to the background that should not be tracked. But if tracked points only appear along the outer edge of the object and the subtraction removes a couple of pixels too much there might not be any corners left to track.

For ASMS and KCF, the scenarios in which they fail are when there are a lot of occlusion or the contrast from the background is low. Distinguishing objects that are small, with low contrast and moving due to noise is not helped by background subtraction. Other methods, such as motion prediction models are probably more helpful in these scenarios.

Background subtraction might be suitable for detecting tracking failure and occlusion. Other tools are then needed to make use of that information to help the tracking (predict where the object ought to reappear and try to redetect when the region contains a moving object again).

Background subtraction could speed up the tracking since some pixels are not taken under consideration. But all trackers studied are already real-time with a good margin and background subtraction itself is slower so this is not a good motivation for using background subtraction in this case.

5.3.1 OPTFLOW

OPTFLOW works well when the target for the tracking has a good amount of salt-and-pepper texture on it. If it is possible to select a region on the object that contains many features without selecting any background that is optimal. If the selected region contains no such features the tracker will not work.

In its naïveté there are a lot of possible paths to make OPTFLOW better and more robust, a problem with the current implementation is how it handles which points to track. At the moment it operates on the principle that most feature points likely belong to the target (this assumption might be wrong, if the target does not have many features it is almost certainly wrong) and that the few points that are on the background will get averaged out and lost when they move out of the image. One idea is to use some clustering algorithm together with the background subtraction and only keep the cluster which is in a region with a high amount of foreground pixels. This would make it possible to get rid of incorrect points and not be as dependent on the correctness of the background subtraction.

From the optical flow calculation we also get the velocity of the feature points, from the knowledge regarding the camera movement it is possible to estimate what the velocity of a background pixel is. By comparing with the estimate of the background velocity it might be possible to filter out points belonging to the background without performing background subtraction.

When a good selection of features has been made it might be desirable to increase the robustness when tracking the features. One way to do that is to perform backward validation. Take the points you found by tracking the features and reverse the time to

track the points backwards to the image where you know the correct location. Points that end up in the wrong location are assumed to be incorrect and therefore discarded.

The performance with regard to speed depends on the resolution of the sequence and the number of feature points that are tracked. The detection step (and thus also redetection in order to add more features during tracking) depends on the size of the tracking target and not the resolution of the entire image. It is also independent with respect to how many features that are to be found. The reason is that OPTFLOW evaluates the entire region for possible corner features and orders them in a list according to how good they are. How many you use is then simply how many feature points you select from that list.

The tracking step depends on the number of features picked in the detection and the resolution of the entire image. By the use of image pyramids the tracking step is quite efficient but it might be possible to increase the speed by only calculating the optical flow on a smaller region of the image. For example restricting it to the region covered by the previous bounding box of the tracked object.

There are a number of parameters in the OPTFLOW tracker that can be set differently. A choice is how often to redetect for new features within the object bounding box. Redetecting often could improve the robustness if done correctly, if the bounding box is in the wrong position when the redetection is done the tracker might get lost completely. Redetection should be performed when it is likely that the bounding box covers the object.

The downside of many parameters is more parameters to tune, which can lead to overfitting to certain scenarios. It is in general desirable to be able to operate under different conditions.

5.3.2 KCF

KCF is shown to be robust and fast on the sequences we tested it on. According to [17] the speed of KCF is directly related to the size of the tracked region.

The tracker is local in that sense that it only evaluates samples in the current image around the previous position of the object. If the object moves very fast in the image, which may also be due to that the camera moves rapidly, no sample contains the object and is valid but the object is still in the image somewhere. This may be of less concern for PTZ cameras since they often have a smooth movement due to the pan-tilt restrictions, and therefore the use of KCF could be appropriate for PTZ tracking.

Information about the movement is available from the system. If used during change of zoom level (not in the scope of this thesis) it must be accounted for that the tracker is not detecting in the whole image.

The implementation we used tracked on HOGs, a more complex feature than raw pixel values, but KCF is also capable of working with raw pixel values. This may be better if the object edges become blurred since large gradient features represent sharp edges. This is connected to the camera movement, objects become more blurred if the camera moves fast. Since a PTZ camera is relatively stable, there should not be too

much blur from motion, and KCF together with HOGs could be suitable to use in PTZ tracking.

5.3.3 ASMS

Looking at the sequences recorded with the PTZ camera the tracking ability of ASMS is hard to put to fault. The speed on the other hand took a large hit when the bounding box (and thus the colour histogram representing the target) contained many different colours. This is probably a problem with the implementation and not the algorithm (bad memory locality when comparing pixels to the histogram), running the tracker on a computer with a better CPU the impact of the more complex object was not as severe.

Analysing the results from evaluating the tracker on the VOT2015 dataset it is possible that ASMS has room for improvement. One change that would be relatively easy to implement and evaluate is a change of feature space. Instead of using colour histograms for the backprojection it might be beneficial to use HOGs. KCF was evaluated in [17] using both raw pixel values and HOG descriptors and was shown to benefit from using HOGs.

Changing feature space might make the tracking more robust in the presence of objects with similar colour distribution with different form. The histogram that is used as a template for finding the target is never updated after initialisation which makes it hard for the tracker to cope with even gradual changes in how the object looks.

5.4 Conclusions

There are in theory cases where a good background subtraction result helps a tracker in keeping the correct track, detecting tracking failures as well as detecting occlusion. Failure detection and occlusion detection were not in the scope of this thesis and the computations spent on modelling the background online to aid tracking seems to be better used in the tracking routine, for example fusion of several complementary tracking algorithms.

If background subtraction is to be done with a moving camera, good alignment of frames is very important. An idea is to use the knowledge of the movement to get an initial guess and then use some form of keypoint matching to compensate for noise and fine-tune the alignment.

5.4.1 Future work

To make it possible to evaluate and get good measures of how well different algorithms perform, more evaluation data are needed. It would be of interest to generate datasets of pan-tilt-zoom video sequences where the pan-tilt-zoom configuration for each frame is attached. Especially construct sequences where this information could help the tracking, such as sequences containing occlusion and where the moving object is similar to the background. Using the toolset from the VOT Challenge with the dataset containing frame offset information might be a good way to automate the testing.

To increase tracking performance in the scenario with a PTZ camera there are some different things that could be explored. The background subtraction proved to be of little help to trackers. What might be better is to use the raw probability map in the subtraction instead. Depending on the probability of being foreground, a pixel should contribute more or less to the tracking.

Instead of using the information about the robot motion to make it possible to perform background subtraction it might be better to use the information to create a motion model for the tracked object. For example, keep track of how many pixels the object has moved between two frames and subtract the motion of the camera to get a better estimate of the true velocity. When guessing starting position one can take into account both how much the camera has moved since last frame and how fast the object is moving (if v_{object} is equal to v_{camera}, do not move the tracking box from reported location).

An obvious way to get better tracking in a known specific scenario is to select an appropriate tracker. In the current state of the field, no single tracker works perfectly in all scenarios. But if you know that the camera will only move smoothly and track objects that move smoothly you could optimise for that. Which scenarios a tracker works well in depends on the feature space used. The choices of feature space and tracker are in a sense disconnected. The feature space is how an image is pre-processed before input to the tracker. Using raw pixels (pixel colour intensities) makes the tracking robust to problems that erase edges but preserve colour (fog, motion blur etc.) but makes the tracking vulnerable to rapid changes in colour (blinking lights). One solution to the colour change problem might be to normalise the images, to give them the same mean colour. Another is to simply pick another feature space, like HOGs. Using HOGs the tracking becomes more robust against illumination changes but gets vulnerable to changes that destroy edges.

To get better tracking the simplest approach might be to pick trackers that are good at different things and combine them in a tracker fusion. Combining ASMS and KCF for example should yield a tracker that is robust both to colour changes and destruction of edges. The hard part of doing tracker fusion is to know when to rely on which tracker. The good thing is that many trackers are designed such that it could be possible to calculate a measure of tracker certainty. ASMS and KCF report a score for several possible positions in each frame. Then to be certain the best scores should be much better than the average scores, otherwise all samples are equally bad which indicates tracking failure. In OPTFLOW the number of tracked features still left on the object could be a measure of tracker certainty.

An extension of tracker fusion in a single video stream is to add more cameras and different sensors in a fully connected system.

Bibliography

[1] Y. Sheikh, O. Javed, and T. Kanade, "Background subtraction for freely moving cameras," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1219–1225, IEEE, 2009.

[2] E. Hayman and J.-O. Eklundh, "Statistical background subtraction for a mobile observer," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 67–74, IEEE, 2003.

[3] A. Elqursh and A. Elgammal, "Online moving camera background subtraction," in *Computer Vision–ECCV 2012*, pp. 228–241, Springer, 2012.

[4] K.-I. Kanatani, "Camera rotation invariance of image characteristics," *Computer vision, graphics, and image processing*, vol. 39, no. 3, pp. 328–354, 1987.

[5] D. Murray and A. Basu, "Motion tracking with an active camera," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, no. 5, pp. 449–459, 1994.

[6] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm computing surveys (CSUR)*, vol. 38, no. 4, p. 13, 2006.

[7] Z. Kim, "Real time object tracking based on dynamic feature grouping with background subtraction," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE, 2008.

[8] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Čehovin, G. Fernandez, T. Vojir, G. Häger, G. Nebehay, *et al.*, "The Visual Object Tracking VOT2015 challenge results," Dec 2015.

[9] W. Hu, T. Tan, L. Wang, and S. Maybank, "A survey on visual surveillance of object motion and behaviors," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 34, no. 3, pp. 334–352, 2004.

[10] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song, "Recent advances and trends in visual tracking: A review," *Neurocomputing*, vol. 74, no. 18, pp. 3823–3831, 2011.

[11] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10, p. 4, 2001.

[12] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *AeroSense'97*, pp. 182–193, International Society for Optics and Photonics, 1997.

[13] M. Piccardi, "Background subtraction techniques: a review," in *Systems, man and cybernetics, 2004 IEEE international conference on*, vol. 4, pp. 3099–3104, IEEE, 2004.

[14] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger, "Review and evaluation of commonly-implemented background subtraction algorithms," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, IEEE, 2008.

[15] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *Computer Vision—ECCV 2000*, pp. 751–767, Springer, 2000.

[16] T. Vojir, J. Noskova, and J. Matas, "Robust scale-adaptive mean-shift for tracking," *Pattern Recognition Letters*, vol. 49, pp. 250–258, 2014.

[17] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2015.

[18] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Čehovin, G. Nebehay, G. Fernandez, T. Vojir, *et al.*, "The visual object tracking vot2013 challenge results," Dec 2013.

[19] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[20] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 593–600, IEEE, 1994.

[21] C. Tomasi and T. Kanade, *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[22] C. Harris and M. Stephens, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, p. 50, Citeseer, 1988.

[23] Y. Cheng, "Mean shift, mode seeking, and clustering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 17, no. 8, pp. 790–799, 1995.

[24] M. J. Swain and D. H. Ballard, "Indexing via color histograms," in *Active Perception and Robot Vision*, pp. 261–273, Springer, 1992.

[25] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the circulant structure of tracking-by-detection with kernels," in *proceedings of the European Conference on Computer Vision*, 2012.

[26] Wikipedia, "Kernel method — wikipedia, the free encyclopedia," 2016. `https://en.wikipedia.org/w/index.php?title=Kernel_method&oldid=709375900`, [Online; accessed 12-May-2016].

[27] L. Čehovin, M. Kristan, and A. Leonardis, "Is my new tracker really better than yours?," in *WACV 2014: IEEE Winter Conference on Applications of Computer Vision*, IEEE, Mar 2014.

[28] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Cehovin, "A novel performance evaluation methodology for single-target trackers," *arXiv preprint arXiv:1503.01313*, 2015.

[29] A. Li, M. Lin, Y. Wu, M.-H. Yang, and S. Yan, "Nus-pro: A new visual tracking challenge," 2015.

[30] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger, "Comparative study of background subtraction algorithms," *Journal of Electronic Imaging*, vol. 19, no. 3, pp. 033003–033003, 2010.

[31] Y. Feng, S. Luo, Y. Tian, S. Deng, and H. Zheng, "Comprehensive analysis and evaluation of background subtraction algorithms for surveillance video," *Sensors & Transducers*, vol. 177, no. 8, p. 163, 2014.