OBJECT LEARNING BY TRACKING AND TRACKING BY DETECTION

Abstract

In this thesis I will explain and analyze a method for online long-term tracking with minimum prior information. 'Online' meaning that no information is used from future frames and it works in near real-time. 'Long-term' meaning that it can, ideally, track across scene cuts, fast camera movements and object disappearances. And 'minimum prior information' that it is a semi-supervised system that works with a single ostensive definition in the first frame. The method will mainly be tested on eye sequences recorded with a mobile phone. The goal is to describe the method and its components and test them under challenging circumstances. I will demonstrate how the framework can indeed build and learn a fairly robust model from a single sample, but also that there are limitations to its capabilities. Especially light changes seem challenging and it has issues with false positives as the model grows. Concluding modifications, extensions and issues will be discussed.

Thesis in Software development, spring 2012

by Morten Granau Høfft - ITU username mgho

Supervised by Dan Witzner Hansen

TABLE OF CONTENTS

OVERVIEW

In this thesis I will investigate an approach to long-term object tracking based on a combination of tracking and classification. The resulting framework allows for easy training of a long-term tracker from a single ostensive definition in a video sequence. In this first part I will talk about:

1) *Why representations are interesting, what focus is and how the thesis is structured.*
2) *General challenges.*
3) *A brief introduction to tracking and classification.*

INTRODUCTION

Last year the electronics manufacturer Foxconn announced that they are expanding their 1.2 million large human workforce, with 1 million robots. These robots are to work by side with humans at the factory floor, performing product specific actions. For this to be possible they must acquire situation-specific representations. In this thesis I will describe a method for learning- and recognizing objects in videos. The method is interesting because it is extremely simple to train and can process video at a high frame rate (15 fps). The main idea is to combine tracking and classification to make both more stable and robust.

Recognition is vital to many applications, from self-driving cars to medical analysis and industrial robots. It is also a difficult problem that has been of scientific interest for many years. I have chosen to focus on visual recognition, as there is much information to be gathered from light alone. After 60 years computer vision is still a very popular research topic and the usages are many. High-powered computers are nowadays readily available and even come in pocket-sized formats and as cameras are getting of higher and higher quality, while the price decreases, the need and possibility for automated image analysis is very much present, whether it is for private, governmental or corporate usage (Hapgood, 2006).

MOTIVATION

Before giving an overview of the outcome of the thesis I will briefly sketch the history and use of representations. Recognition and mental representations seems to me a cornerstone for an intelligent being, artificial as well as biological. The ability to recognize allows us to collect information over multiple encounters and, more importantly, use it to guide actions.

Representations and their usage have since the birth of AI been a key element. Perception and construction of representations were for many years given less, if any, attention. Since then representations (or representationalism as the mindset has come to be known as) have been under attack several times. Most noticeable by Hubert Dreyfus in his seminal work: "What Computers Still Can't Do" (1992). But even Dreyfus accepts some form of representations (Dreyfus, Why Heideggerian AI Failed and How Fixing It Would Require Making It More Heideggerian, 2008), which seems to be very close to neural networks as they are trained in e.g. Q-learning. That is, a classifier that has a build-in propensity for generalizing and that are in a constant state of flux or

dialog with the surrounding world; where every sniff and whiff modifies one's conceptions. In good old-fashioned AI, GOFAI, as John Haugeland has coined it (Haugeland, 1985), representations and symbolic manipulation is the main focus, but is also what drives the whole process of Rodney Brooks, anti-symbolic robots (Brooks, 1991). Brooks' subsumption architecture simply couples perception, representation and reaction in a much tighter loop, making use of simpler representation and less reasoning; a strategy that worked really well. Almost every aspect of intelligence seems to depend on representations and recognition; whether it is chess playing or navigation. So it can be argued that representations is one of the most important abilities of intelligent beings (Millikan, 2000)[1]. From a practical point of view, machine representations and recognition is useful in numerous applications.

- Industrial robots: production line, visual quality control and inspection. E.g. automated pancake chef.
- Navigation: e.g. autonomous vehicles.
- Event detection: e.g. counting people in line or in the room. Detecting accidents or law violations.
- Interaction: allowing various forms of machine interaction by – for example – hand gestures.
- Medical analysis: analyzing large databases of image data.
- Image search: e.g. ask for all scenes with Catherine Deneuve in them.

Vision is, of course, only one among many ways of recognizing. Just take sugar: to identify sugar from salt I could e.g. read the label, taste it or give it to a chemist. Nevertheless this thesis will focus on the visual aspects of recognition.

THESIS OUTCOME

I have chosen to look at online long-term tracking and learning with a minimum of prior information. To specify the terms:

- Online meaning that video can be processed without perceivable delay (real-time). This is not always a requirement, for many applications post-processing is adequate, but for some purposes online-processing is a requirement. An example could be a self-driving vehicle that need to track and recognize objects in real-time.
- Long-term trackers recognize an object from frame to frame while handling object disappearance, occlusion and object transformations. This is in contrast to classic trackers that are short-term. This is

[1] Though Millikan uses the term 'concepts', she has elsewhere in her oeuvre used 'representations' and I will argue that for, this purpose, they can be used interchangeably.

especially useful in scenarios where the object undergoes large transformations and/or temporary disappears. An example could be surveillance were a person need to be tracked across cameras, occlusion, disappearance and multiple angles.

- Minimum prior information in contrast to the many classifiers and trackers that need a large database of object samples to work. Collecting these samples can be time-consuming and building a classifier based on these samples often need to be done offline. And if we do not know beforehand what to recognize neither data collection nor training is an option. In these cases it is useful to be able to start with a single example and learn iteratively and online. An example could be gaming where we need to recognize the individual players after first encounter.

Combining tracking and classification seems promising. The work in this thesis is a simplified version of an idea presented by Kalal et al. (2010) and the result is a long-term tracker that Tracks, Learns and Detects (TLD framework). The framework is based on the idea that tracking and classification complement each other:

Tracker: A tracker is an algorithm that can estimate the trajectory of a chosen object in a video sequence. Most trackers are short-term trackers in the sense that they will loose track once the object is out of sight and generally works best over short periods of time. If we had a classifier that could recognize the object once in a while, we could use this classifier to detect and restart the tracker when object trajectory was lost.

Classifier: An object classifier is an algorithm that can recognize either a class or a unique substance. For a classifier to know what to classify it needs to by told or otherwise learn it. This is often done using examples: is and is-not. Creating these is time consuming and not necessarily something that can be done in advance. If only we had a tracker – it could then be used to gather relevant samples to train our classifier.

TLD: The TLD framework fulfills these two needs by creating a framework for interaction. Given a video sequence, and an object of interest, the tracker starts following it. While doing so the tracker teaches the classifier how the object looks throughout the trajectory. When the track is cold, the newly trained classifier steps in, detects the object and we can start over. Ideally this means that the classifier gets better and better over time and that we only need to make one manual object demarcation for the algorithm to work.

The proposed framework will be described in detail and tested. Handheld recordings of eyes are used for tests. Recordings made with a mobile phone are challenging, as they can be noisy, blurred and angled and light change often and much. Further more, eyes are a difficult task as they undergo large appearance variations due to having curvature and being flexible and movable (more on that later). Eyes have been chosen as focus area since they seemed within the reach of the framework while still being a challenge. Besides being a generic challenging object from which we can learn something general, it is also useful for practical applications such as user interfaces.

Figure 1 - Taking the eye as an example we see that long-term tracking is a challenging task as the object drastically change appearance over time.

STRUCTURE

In the remainder of the overview, I will give a description and some examples of the challenges involved in recognition and long-term tracking. This will give us an idea of the challenges ahead and what to test for. Thereafter, I will present a brief overview of the large corpus of previous works.

The second part will describe the method in details: how are short-term tracking and long-term tracking combined in the proposed framework. This includes a thorough description of the two framework components: the short-term tracker and classifier.

The third part is tests of the two components and the framework as a whole. The components will be tested on mainly synthetic material, whereas the framework will be tested on actual video recordings of various objects and eyes.

The concluding chapter will give a summary of what we have seen and learned and present some ideas for modifications and extensions.

CHALLENGES

This chapter considers the challenges involved in recognition and tracking. Long-term tracking is — as so many other sub disciplines within computer vision — a difficult task. There might be: background clutter, objects of the

same type, fast camera movements, fast and irregular object movements, objects deformations from rotation or movable parts, changing light conditions, low frame rate, signal noise, odd irregular shapes, occlusion from other movable objects, itself or the static scene, scaling, texture transformations, transparent or reflecting surfaces, data loss in conversion from 3d to 2d and it might even be completely out of sight for a while only to reappear. On top of it, we often need to do it real-time and the slower the algorithm gets, the more difficult the task, as the frame rate drops further.

Nevertheless, we humans do this with what feels like little effort, but in cases where we need to concentrate a bit more on say, a ball being passed between players, we fail at noticing much else. In fact experiments show that we probably will not even notice if the ball disappears behind a roaring gorilla (Chabris, 2010). So even to humans, tracking seems to be a somewhat demanding task. With all these usages and challenges it is almost as obvious as sad that no silver bullet has been found. On the bright side there a myriad of approaches, each with its qualities. Before looking at some of the existing solution strategies, I will describe the challenges in more detail.

Figure 2 - Still from Chabris and Simons original awareness experiment. Subjects were asked to count ball passings. The strain of tracking the balls led them to miss the fact that a man dressed as a gorilla entered the frame and thumped his chest.

EXAMPLES

To understand the challenges we face when building trackers and classifiers, I will next describe and give examples of a subset of them. The examples will all be of eyes, as these will receive most attention in the thesis.

Object changes

The substance of interest might change pose, color, haircut, pattern, form etc. This poses a real challenge for many objects: e.g. scissors (pose change) and television (pattern change) and eyes. If we have learned to recognize an open eye looking straight ahead, it is not at all obvious that it is the same eye when closed or looking to the side. Learning that this transformation does not change the object' essence is a serious challenge that can lead to complete tracking and recognition failure. Even something as simple as a missing beard can lead to recognition failure for children (Millikan, 2000).

Figure 3 – Examples of object changes. Recognizing that they are all samples of the same eye is no easy task.

Angle

An object will often look very different from different angles due to perspective, self-occlusion and affine transformations. Just think of a head seen in profile versus en face and from the back. Never the less we are confident at recognizing each other from most angles and distances.

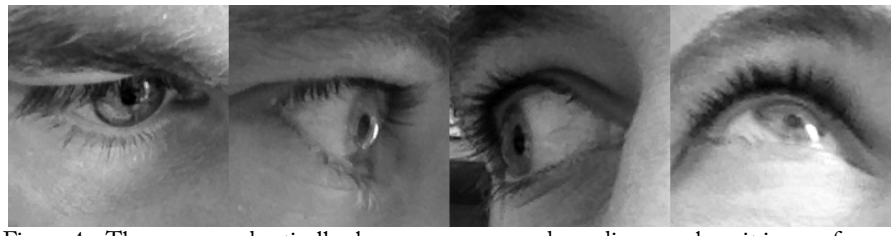


Figure 4 - The same eye drastically changes appearance depending on where it is seen from.

Occlusion

Whether it is partly or full, brief or long, occlusion is a key element in tracking. Objects are often hidden, partly of full, at some point in our tracking of them – behind another object, themselves or simply out of view. Recognizing objects even though they are not fully visible is further a very attractive quality (just think of finding something in the refrigerator without this ability). And for a long-term tracker to work we need to be able to handle full occlusion.

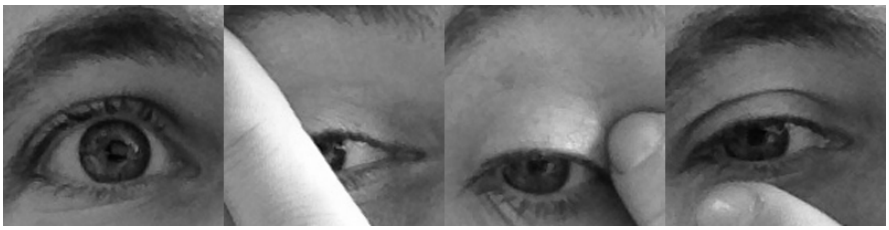


Figure 5 - Occlusion. Tracking through occlusion as well as recognizing occluded objects, is important but difficult.

Light

Light is yet another factor in a long chain of challenges to tracking and classification. Object appearance can change immensely depending on lighting. Color may change and even features that where easily detectable before may completely disappear under the 'wrong' illumination or exposure. Moving the light source may even be mistaken for object movements. Therefore most approaches try to extract as viewpoint and illumination independent features as possible.

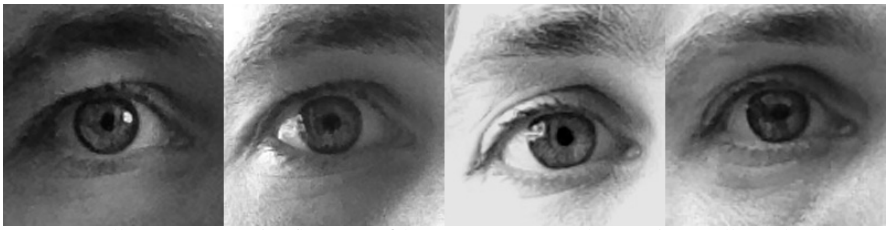


Figure 6 – Illumination and reflections. Objects appearance change drastically with illumination.

Position and scale

A classifier that is robust to position and scale changes is preferable. When searching for an object in an image we will often query sub patches. Doing so at all scales and all positions is very time consuming. If the classifier is robust to these translations we can scan with a much cruder resolution. Further these transformations can be seen as primitive versions of the many other object transformations.

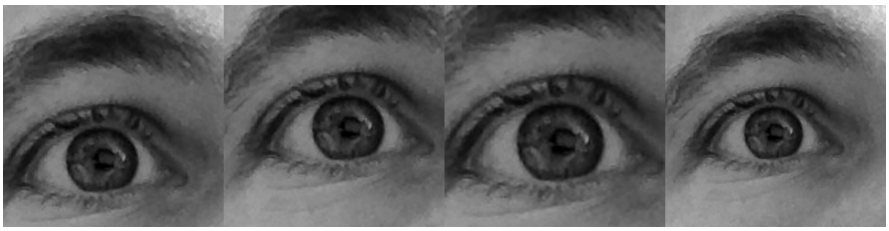


Figure 7 - Position and scale. Being able to recognize the object even though it is slightly moved or scaled is very useful.

Degradation

The quality of the image is always less than what one could dream of. There might be noise due to low light or camera interference. Motion blur due to fast movements and to long exposure time. And the object might have low resolution due to distance or video quality. And so details that are normally used to recognize can become inaccessible.

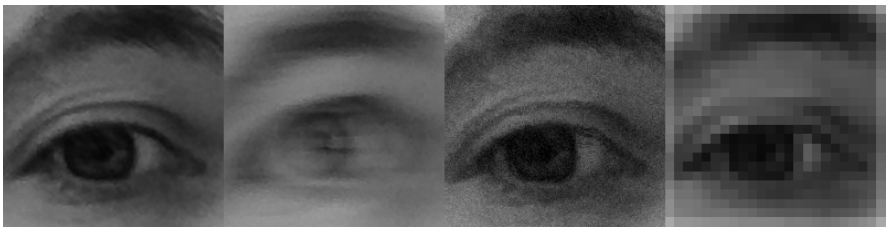


Figure 8 - The image patch can be blurred, noisy or of low resolution. This complicates recognition further.

Similarity

We have seen that the same object can look extremely different under different circumstances, and yet we should be able to recognize it. We further need to separate it from all the other objects that look alike – either because they belong to the same class or simply because they have similar traits. It might not even be another object, but a combination of objects seen with a specific cropping. Often these interclass similarities are greater than those intraclass, making the partitioning very difficult.

Figure 9 - interclass similarities. These four images have, in many ways, more in common than the left image has with the above examples. And yet these four are all different substances. From left to right: Same eye as above, car salesman, butterfly and makeup model.

Other

Further there are issues as atmospheric distortions (e.g. heat or fog), seeing the object through diverse media (e.g. glass or water instead of air), exposure time, white balance, reflections and real-time processing requirements.

TRACKERS AND CLASSIFIERS

We have now seen some of the challenges involved in tracking and classification, but have not yet looked at tracking and classification in itself. I

will here give a brief introduction to both and mention previous work that are relevant in the context of the method explored in the thesis.

- General notes on tracking.
- Previously on long-term tracking.
- General notes on classification.
- Previously on semi-supervised classification.

Of course this is but a crude introduction to the vast amount of previous work, but will hopefully set the scene and give an overview of the diversity of the strategies involved. Thereafter the implemented method will be described.

Tracking

> *"In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. In other words, a tracker assigns consistent labels to the tracked objects in different frames of a video. Additionally, depending on the tracking domain, a tracker can also provide object-centric information, such as orientation, area, or shape of an object." (Yilmaz, 2006).*

Tracking and classification is, in some sense, the same ability; namely the ability to lock on to a substance and recognizing it on future encounters, whether it is in 1 second or in 1 week. The main difference is that the tracker has access to more information, namely the previous location. The tracker further has to describe the object trajectory and it often needs to process video in real-time. Luckily it also has more information to work with than the classic classifier that works on isolated images. This means that the tracker might make do with a cruder — and potentially faster and/or simpler — recognition strategy.

Representation

A natural first choice would be to decide on object representation. What do we need to know about our objects location? Is it enough to know the position of some point on the object or do we need the center, specific points, the bounding box, enclosing ellipse, contour, object skeleton or the silhouette? One can imagine multiple ways and degrees of object demarcation depending on subsequent usage and each provides different opportunities and challenges — computational and otherwise. These shape-related ways of representation can also be used to identify our object. Does an image patch, for instance, have a skeleton similar to the sought for model. We can also choose to identify our model by its appearance (better described as surface properties). This could for example be a collection of features such as texture, edges or color, or it could be an image template. The approach used in the proposed tracker is a bounding box (see figure 10-c).

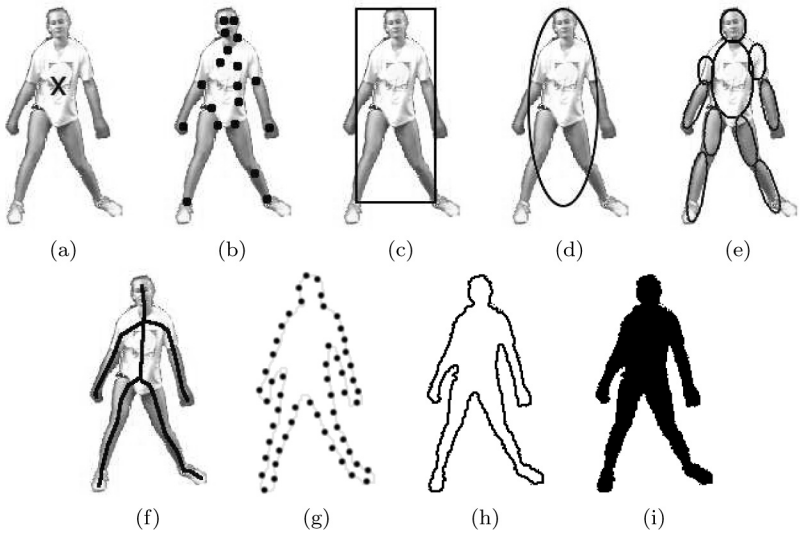


Figure 10 - Dependent on usage we need to choose a tracking representation. Approach c will be used in the implemented method. This approach works best with rigid objects. Image from (Yilmaz, 2006).

Adaptive

A tracker can either be adaptive or more conservative. The adaptive tracker is suited for tracking objects that change appearances (e.g. turning or posture), but vulnerable to drifting and occlusion. If the expected appearance of the tracked substance is updated at every frame, a small tracking error can result in us tracking something completely different. The underlying assumption in adaptive trackers is that every update is correct. On the other hand we have trackers that use a predefined one-view model (e.g. template matching), who are robust to drifting, but looses track if the tracked objects turns away from the camera or changes appearance in any way (Yilmaz, 2006).

PREVIOUSLY ON LONG-TERM TRACKING

Standard short-term trackers perform frame-to-frame tracking and more or less ignore disappearance. Instead they focus on reliability and lifetime, avoiding drift and handling partial occlusion. But for long-term tracking we need to handle scene cuts and disappearances.

Grabner et al. has developed a method for tracking the hidden object by tracking what they calls 'supporters' (Grabner, Matas, Gool, & Cattin, 2010). Cerman does the same, but calls them companions and only work with a static background (Cerman, Hlavac, & Matas, 2009). The idea is to detect other areas that covary with the object. When the object is lost, we can then estimate its position based on these companions/supporters. This, of course, only works if there is any such trustworthy covariance. And scene cuts are still not handled.

Another approach to long-term tracking is to detect the object once it reappears. An increasingly popular method is tracking-by-detection.

> *"Such approaches involve the continuous application of a detection algorithm in individual frames and the association of detections across frames. The main challenges when using an object detector for tracking are that the resulting output is unreliable and sparse, i.e., detectors only deliver a discrete set of responses and usually yield false positives and missing detections".(Breitenstein, Reichlin, Leibe, Koller-Meier, & Gool, 2009)*

An example is Lepetit (Lepetit, Lagger, & Fua, 2005) who, like I, use randomized trees for keypoint recognition. But their classifier is trained offline. For real-time applications AdaBoost is often used (Viola, 2001). But AdaBoost is also trained offline. However, there do however exist an online adaption (Grabner & Bischof, 2006), but in its current form it discards old features adapting to the new samples, making it less suitable for our purpose. Breitenstein combines an offline-trained model with an online instance specific model, but it is designed for pedestrians and still relies on offline training (Breitenstein, Reichlin, Leibe, Koller-Meier, & Gool, 2009). If we know the scene layout and object beforehand, very efficient detector-trackers can be created (Taylor, Rosten, & Drummond, 2009), but this is not always the case.

Tracking and classification can also be combined(Andriluka, Roth, & Schiele, 2008)(Ramanan, Forsyth, & Zisserman, 2005), but the detectors in these two examples are trained before start and thus cannot be used when the object is unknown. Besides, they are both designed for classes and not particulars. Training of the classifier are in many cases done using large manually labeled training sets or – in case of Lepetit (Lepetit, Lagger, & Fua, 2005) – done by warping the patches (as we shall see, the implemented method does this as well, but also expand the model at run-time) or using variants of semi-supervised method as we will see in when discussing classification.

Classification

Object recognition has for many years been approached geometrically. The attempt has been to describe objects as idealized geometric structures and matching these with manually constructed models of our target object. Doing this requires extracting edges (taking into account shadows, uniform lightning, textures etc.), grouping these and translating them to geometric primitives in a 3D-space. When this is accomplished it can be a very useful tool for recognizing objects. But not all objects are recognized best by geometry and worse yet, extracting these idealized geometric primitives — geons as Biederman coined them (Biederman, 1987) — is no easy task.

This approach was gradually replaced by the intensity appearance-based in the late nineties. One useful thing about the appearance-based approach is that it does not need complex object segmentation and reconstruction. Mundy writes:

> *"One way to look at the current state of object recognition research is that the four decade dependence on step edge detection for the construction of object features has been broken. Step edge boundaries are still useful in forming an object description where the object surface is bland and free of surface markings. But, for a large fraction of object surfaces and textures, affine patch features can be reliably detected without having to confront the difficult perceptual grouping problems that are required to form purely geometric boundary descriptions from edges." (Mundy, 2006)*

Unlike the geometric approach, the appearance-based classifier is typically learned and not predefined. But how does it learn?

Learning

'Saving' an object — or more generally a substance — for future recognition, is done by extracting stable features; whether it be geons, colors or corners. But choosing these features is no trivial task. Ideally we are able to extract unique features for the individual substance and ontological class. And these features are ideally observable in all conditions. But the world is not so. The features of a substance are more often than not hidden. This is why we need to remember and recognize substances in the first place. For example: when I first approached a cat I did not know that it had claws and were willing to use them. I soon learned that this particular substance did not respond well when poked. I got scratched, scared and decided to remember this type of substances. Had the features of the cat been clearly visible at all times (that it had claws and knew how to use them), I wouldn't have to remember anything. I would simply extract the potentially dangerous features and go somewhere else. But the claws are hidden and so is its temper. So I need to remember this dreadful creature for future encounters, but how to recognize it? What are its stable features? Which can I expect to encounter on my next meeting? Sometimes location is crucial, other times not. Sometimes color, size, form, posture, name, time a day etc. Most likely it is a combination and depending on circumstances. A good way to extract these stable features is to find similarities between examples, but this is a circle as identifying samples as being of the type 'cat' is just the type of ability we were trying to learn in the first place (Millikan, 2000). This vicious circle is the problem of learning. In case of machine learning, this problem is typically solved using a training set created by a trainer: 'is' and 'is not' combined with assumptions about the order of things.

Supervised and unsupervised learning

Manually defining a failsafe way of recognizing a given object or class through eidetic variation is difficult, if not impossible. This goes for both the geometric and the appearance-based approach, but even more so for the latter. When using intensity appearance it is impossible to define manually which pixels should have what intensity in relation to which other. A generic algorithm is therefore often used in combination with a labeled training set (a list of image patches and a label telling if it is an instance of our object or not). If we train entirely using a labeled training set we have supervised training. The idea behind a labeled training set is to have the classifier analyze the pictures of object/non-object and from this generate a strategy for separating the two classes. Ideally this strategy will work when confronted with a before unseen sample. The difficulty of training from samples can be illustrated by the classic story of the military' attempt to create a system that could recognize tanks amid foliage.

> *"Scientists fed pictures into a neural network of trees with and without tanks parked beneath them. At first, they had stunning success — the machine had a 100 percent detection rate. But when they tried reproducing the results with new data, the ANN failed. The computer hadn't learned to detect tanks at all. Instead, it had focused on the color of the sky to determine whether tanks were present because the test photos had been taken on different days. In the pictures with the tanks, the sky was cloudy; in the pictures without tanks, the sky was bright blue. The network had learned to recognize the difference in the weather."(Perera, 2008)*

Apocryphal or not, the story illustrates one of the challenges (and the interest in recognition): carefully selecting training material is paramount for success.

In unsupervised training, we present our training set, but with no labels attached. It is then the job of the algorithm to segment the data into different classes based on some clustering strategy. This tabula rasa approach sounds great, as labeling is notoriously difficult and time consuming. It is difficult since the training data have to be diverse and representative (as demonstrated in above example). And it is time consuming as many algorithms needs thousands of examples to generate a sufficiently precise classifier (Viola et al. (2001) uses 5000 different face images to create their classifier). The better the examples, the fewer needed. But automatic clustering is not without problems either. One could argue that our object categories were based on the affordances the objects provide and that a disembodied machine is left helpless organizing the data to our satisfaction. Instead it clusters entirely on similarities in a meaningless feature space. Making the, already, important choice of features even more vital. The same could partly be said for semi-supervised classifiers, but not to the same extent (Zhu, 2008).

Semi-supervised learning strikes a balance. The idea is to have a large unlabeled dataset and a few labeled examples. The examples will serve as pointers to what we want and the unlabelled dataset will then be used to clarify it automatically. This strikes me as extremely practically and more biologically plausible than either of the two extremes. Experiments also indicate that we as humans make use of semi-supervised learning (Zhu, 2008). This is also what will be used in this thesis, but before presenting the method, we will look at previous work with semi-supervised classifiers.

PREVIOUSLY ON SEMI-SUPERVISED LEARNING

Semi-supervised can be useful since large datasets are often needed to create sufficiently robust classifiers and unlabeled data is often easy to come by, whereas domain experts are needed to label the training set and this process is time-consuming. Further hand labeling is only possible when building our classifier offline. Several elegant strategies have been devised to learn from partially labeled datasets.

Self-training

Self-training is a wrapper method. Given a classifier, a large unlabeled dataset and a few labeled samples we can expand our training data iteratively. First we train our classifier on the already labeled examples. We then classify the entire dataset of unlabeled samples. The samples that are most confidently identified are added to the set of labeled samples. The classifier is then retrained on the new – and larger – set of labeled samples. This process is iterated any number of times. The obvious danger is the risk of reinforcing a classification mistake. There exist various modifications of the algorithm that attempts to avoid this spiral degeneration of the classifier (e.g. by unlearning) (Zhu, 2008). It has been used with success on various areas outside image recognition and it has been demonstrated that it can perform very well on image classifications tasks as well (Rosenberg, Hebert, & Schneiderman, 2005).

Co-training

Co-training assumes that the features can be split into two sets and that each set is sufficient to train a classifier. Two classifiers are then trained using the two feature sets respectively. Most likely the two classifiers are now able to classify different samples of the unlabeled data. The samples that are most confidently classified are used as labeled data to train the other classifier. This process is repeated until some criterion is met. It is of course important that there are features enough to generate two decent classifiers. And these features should be independent since the exchange of knowledge between classifiers

does not work well if they identify the same samples (then it is simply self-training with fewer changes).

One way to ensure independence is to use different views or modalities; e.g. sound and image. This combination has been used to identify users based on gestures and voice (Christoudias, Urtasun, Kapoor, & Darrell, 2009), and vehicles (Godec, Leistner, Bischof, Starzacher, & Rinner, 2010). If sufficiently independent feature views cannot be obtained there might be other ways. Zhou suggests co-training, not with different views, but with different types of classifiers. If the classifiers differ sufficiently in classification biases this strategy works well (Zhou & Goldman, 2004). Zhou also introduces tri-learning where, instead of using the most confidently identified samples to train the other classifier, a majority vote among the three classifiers is used (Zhou & Li, 2005). This also allows for usage with binary classifiers. Tri-learning could be said to be a multi-view classifier. Multi-view classifiers are a general term for classifiers that train different classifiers on the same training set. Some voting procedure is invoked to create the final result.

Cluster-and-Label and others

Another approach is to 'simply' cluster the whole dataset – labeled as well as unlabeled – and use the labeled samples as representative of the clusters. The success of this approach is highly dependent on the feature space and clustering.

Besides these there are Transductive Support Vector Machines (an extension of SVM), Expectation Maximization and Graph classification. All the above-mentioned techniques have the advantage of being very general strategies for making use of unlabeled data. They have been used to classify words, sentences, audio, images and video. This versatility is of course, commendable, but more specialized ways of exploiting unlabeled data might be possible when focusing on a specific media and application.

Learning by constraints

One way to solve the conundrum of the circle described in the beginning of this chapter is to ask someone who knows (whether it be one's mother, an ornithologist, or a programmer). This is what corresponds to previous mentioned training set used in supervised and semi-supervised learning. But as previously mentioned, creating this training set is laborious. And it is difficult to make it representative of the substance we try to circumscribe. Learning from a single encounter is an extreme version of semi-supervised learning.

Another way to acquire stable features would be to trust once ability to track. Tracking the substance over time allows me to collect more information, markedly expanding my knowledge and thereby chances of recognizing the

substance. I can poke, squeeze, smell, taste or — in the case of recognition and learning from video sequences alone — I can see it from different angles in different light conditions etc. I therefore believe that tracking is a very important area for machine learning, as it provides a way in to a seemingly vicious circle. Providing the means to create precise classifiers that can adapt to circumstances that are difficult, if not impossible, to train for beforehand. Further the training material collected this way is very relevant, as it originates in the actual context. To do this a framework that allows tracker and classifier to communicate is required. This is what will be presented in the following section.

OUTCOME

So far we have mainly looked at the challenges involved in – and strategies evolved for – object tracking, learning and recognition. We have only briefly touched the TLD framework. In this section I will give a more thorough description of the framework and its two components: the tracker and the classifier.

TLD FRAMEWORK

Tracking presents a way into semi-supervised learning. As discussed there are many semi-supervised techniques. One way is to use the spatio-temporal constraints inherent in videos. In a video we have a large amount of frames that can serve as unlabeled data. If we labeled a few frames and then generated a classifier from the labeled frame and the abundance of unlabeled frames, we would have a classic semi-supervised system. But the video sequence contains more information than separate samples. They are spatially and temporally related. The idea is to look at the video as a constrained set of training samples. The constraints considered in current approach are continuity in space and time and uniqueness of the object. Other constraints might exist in a specific context or other domains. These constraints allow us to utilize the unlabeled dataset in other ways.

We can use the structural constraints that exists in videos to help us label our unlabeled samples and thus train our classifier. Using a perfect tracker, we simply have to demarcate our object in the frame where it first appears. The tracker will then collect examples of whatever way our object shows itself. If we further assume that we are interested in a unique object, we know that everything else in the video is clutter. We can thus also use surrounding patches as training materiel. So ideally the object is pointed out (someway or another) and then tracked. This tracking provides perfect information on what the object looks like under different conditions and — just as important — what it does not look like.

But no such thing as a perfect tracker exists, but a decent one might also do if the classifier we use is not overly susceptible to noise. Whenever we trust that the tracker has indeed tracked the correct patch we can add this patch as a positive sample to our training set. And remaining image patches can be added as negative samples. Adding a positively labeled patch can be thought of as growing our classifier, and adding a negative as pruning. This leaves us with two questions: “when to grow?” and “how to initialize?”. This will be discussed next.

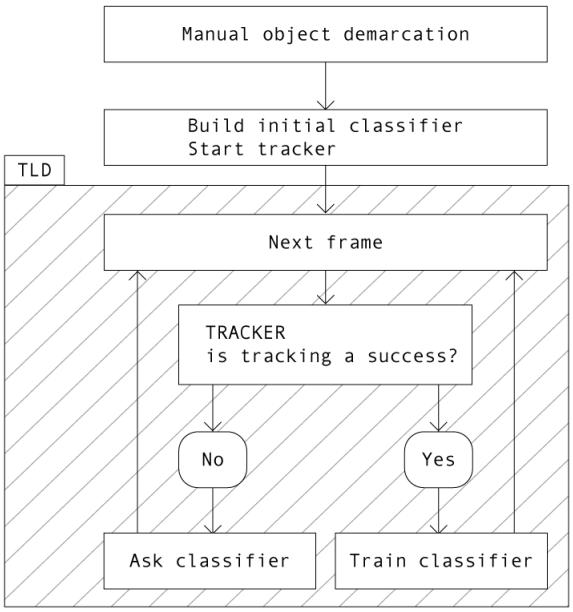


Figure 11 - A first sketch of the TLD framework.

WHEN TO GROW

If we add all patches suggested by the tracker we might risk adding clutter if the tracker has drifted or the object is occluded; we want to minimize that risk. There are several options.

Absolute A cautious strategy could be to only add patches that are similar to the starting patch, given some similarity measure. This can sound self-referring: if we already have a trustworthy way of measuring similarity, then why build a classifier? The idea would be, that we need a less reliable classifier when evaluating the similarity, since it is unlikely that tracker and our coarse classifier agrees if it is not the desired object. The coarse classifier would then define how much transformation was allowed. If the coarse classifier is to remain sufficiently reliable, it will most likely mean that little variation is allowed into our labeled training set.

Adaptive Another strategy — that would allow for more object variation — could be to add patches that are similar to the previous patch. If there is too high a difference between two frames we will stop adding to our labeled training set. This would work in the sense that we would get a more diverse training set, but with the increased risk of adding erroneously labeled patches. Slow drifting would be accepted.

Loop It is improbable that tracker and classifier agrees on object position if tracker has drifted or been occluded. This observation is useful. Instead of only adding similar patches, we can add all patches between our last trusted patch and our next trusted patch. That is: we decide that this patch is an example of our object. We then track the object. If the tracked patch (within some given temporal interval) returns to a familiar appearance, we accept all intermediate frames. The idea is that it is probably the same object we have been tracking all along; we simply saw it temporarily from another angle or in different lightning. The classifier we use to judge whether we are back in a familiar object state could be any number of classifiers. Kalal et al. (2009) suggest using the classifier itself or normalized cross correlation. In essence we just want to validate that it indeed is the correct object we have tracked.

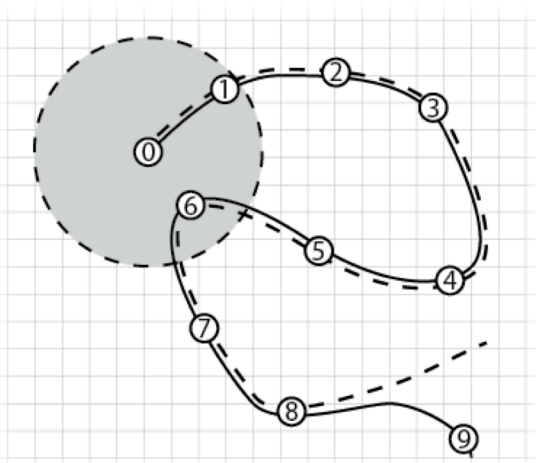


Figure 12 - Feature space with full line showing the tracked patches and dashed line the actual object transformations. Circles are patch projections into our feature space. Gray circle shows the coarse classifiers object demarcation. The Absolute labeling strategy will add {0,1,6}. Adaptive will add {0-9}, falsely labeling 9 as the object in this case. Requiring more similarity between frames could solve this. But then it would only add {0-3}. Using the Loop method we would add {0-6}, ignoring 7 and 8 even though they are correctly tracked.

INITIALIZING THE TRACKER

Sooner or later our tracker, or any tracker for that matter, will loose track. Either because the object disappears from sight, there is too much noise, drift or, in the case of edited videos, there is a scene cut. When that happens our semi-supervised learner will – hopefully – stop learning if it is not initialized again. This can either be done manually (as when first started) or — and more interestingly — automatically. If the newly built classifier is sufficiently trustworthy we can use it as detector and initialize the tracker anew. Alternatively, we can use a harsher classifier, using only our manually labeled frame (e.g. template matching). Doing so does not guarantee that the object is

detected first time it reappears, but when detected, our system will continue learning. In an offline scenario we can iterate over the video several times and, if using the detector itself to initialize, we might detect the object earlier on the subsequent iterations. Our classifier might even adjust our tracker if a promising candidate is detected elsewhere in the image, getting it back on track.

Automated initialization means that our tracker shifts seamlessly from adaptive tracking to tracking-by-detection and back again. Doing so ideally results in a tracker that can handle occlusion and other occasional tracking failures through classifier initialization. And at the same time we create a robust classifier from a single training image by adding multiple views of the object through unsupervised training by tracking.

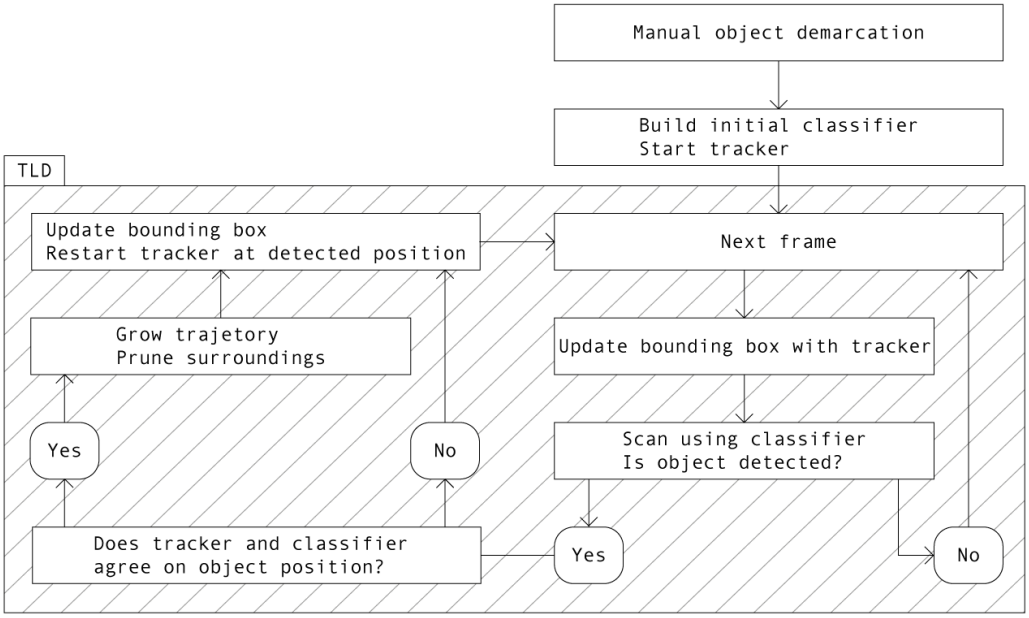


Figure 13 -Diagram of the TLD framework. The user initializes the framework by choosing an area of interest. The framework will then attempt to track the object and simultaneously build an object classifier. The classifier is used to rectify the tracker.

EXAMPLE DESCRIPTION

We choose our object in the first frame and create training samples from this image. The chosen area is a positive sample, the remaining image patches is negative samples. This is used to create a classifier. We then start tracking the object. At every frame we ask the classifier to scan the entire image. If the classifier detects a patch we either grow or initialize: if tracker and classifier agrees we grow the trajectory. If they disagree we trust our classifier and initialize the tracker. If the tracker does not detect anything we continue tracking.

PARAMETERS

The framework allows for any tracker and any classifier as long as it learns from samples. These are the largest parameters. Besides there is the similarity measure for deciding when to add samples to the training set and the threshold for when to initialize. This can either be done using the trained classifier or we can use a separate classifier (i.e. template matching). Besides these two thresholds there are several possible small modifications. Loosening growth, pruning more, scanning with finer resolution, pruning more often etc. Before we can go any farther we need a tracker and a classifier.

TRACKER

The dynamic structure between the tracker and the classifier allows for any tracker to be used. But naturally, the better the tracker, the better the result, since the tracker generates the training data for subsequent use by the classifier. I decided to use a modified version of the median flow tracker (Matas & Vojir, 2011). The tracker is based on optical motion flow, which maps points in the previous image to points in the present image. This is done using Lucas Kanade tracking (Lucas, 1981)(Bouguet, 2000). The Median flow tracker has three parts and these will be described separately: 1) point flow, 2) selection of points and 3) interpretation of tracked points.

Figure 14 - point flow of points between two images. White lines denote where the point is estimated to be in the preceding image - black where it came from.

POINT FLOW

The point flow estimation by Lucas and Kanade is a way to track the apparent motion of image brightness between two images. Ideally this creates correspondences between real world object points and pixels. An example could be to track the pupil of an eye in a video sequence. The technique relies on three assumptions:

1) Brightness constancy. There is no change in an object point's brightness as it (possibly) moves from frame to frame.
2) Spatial coherence. Neighboring pixels belong to the same object and move similarly to the tracked point.

Small movements. The position of object points change slowly relative to the frame rate. The more frames per second, the more correct is this assumption.

Video representation

We can think of our video as a function that to any given x and v coordinate time t returns the image intensity at that point in time.

1) $I(x,y,t)$, intensity function for video

Figure 15 -The video can be described as a function with three input parameters - x, y and t - that outputs a grayscale intensity corresponding to that pixel point at that time or frame number (x, y is pixel coordinates and t is time or frame number).

Goal representation

Given a point of interest our goal is to find the functions that to any given time t can tell us the location (x,y) of the point. That is, we are interested in the functions $v_x(t)$ and $v_y(t)$ that gives us the x-coordinate and y-coordinate of our interest point to time t.

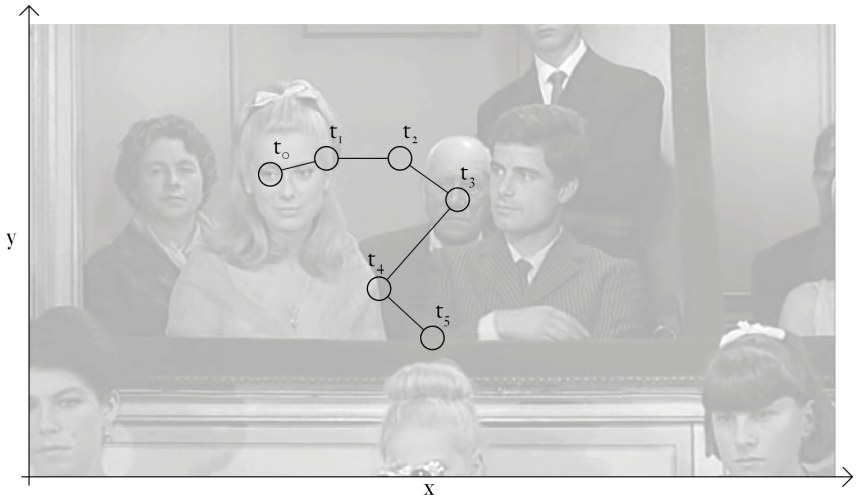


Figure 16 - Ideally we find the goal functions ($v_x(t)$ and $v_y(t)$) that output the x- and y-coordinates of the tracked point for any given time *t*. The image shows the intensity values for t_0. The eye will in the next frame have moved to position t_1.

Searching for the goal functions

The goal functions are unknown, but we can say something about them by using our assumptions. Using assumption 1 we know that if we insert the goal functions $x = v_x(t)$ and $y = v_y(t)$ in the intensity function *I*, we get a constant function (of t). We know this since the assumptions says that a point retains its brightness over short periods of time. In other words: Catherine Deneuve' pupil will also be black in the next frame.

2) $I(v_x(t), v_y(t), t)$ *is constant*

We now know a little more, namely that for all t in function 2 we get the same value. This entails that if we take the derivative with respect to t, we get zero, since there is no slope.

3) $$\frac{d}{dt}(I(v_x(t), v_y(t), t)) = 0$$

Using the chain rule we can rewrite this as

4) $$\frac{\partial I}{\partial x} v_x'(t) + \frac{\partial I}{\partial y} v_y'(t) + \frac{\partial I}{\partial t} 1 = 0$$

We now have a new equation expressing something about the unknown functions $v_x(t)$ and $v_y(t)$ and their relation to the video intensity function *I*. This is a single differential equation in the two unknown goal functions $v_x(t)$ and $v_y(t)$. This, it turns out, is useful. Unfortunately one equation with two unknown functions (as in eq. 4) cannot be solved. But we can get further if we look at the particular instance of time t_0 we are interested in. In this frame we know the outcome of our goal functions – namely the initial location

$(x_0,y_0) = (v_x(t_0),v_y(t_0))$ of our interest point (e.g. the pupil). It turns out (see below) that we can use this information to calculate the slopes $v_x'(t_0)$ and $v_y'(t_0)$.

Since we do not have an actual mathematical function $I(x,y,t)$ from which to calculate the derivative – instead we have the image values for $I(x,y,t_0)$, $I(x,y,t_1)\ldots$ – we need to use these discrete values to calculate an approximated slope. So the derivatives becomes an approximation based on lookups. To get the slope of I with respect to x, we lock y and t to our interest point coordinates and vary x slightly. Defining our interest point as (x_0,y_0,t_0) we see that

$$a_0 = \frac{I(x_0 + x,y_0,t_0) - I(x_0,y_0,t_0)}{(x_0 + x) - x_0} \rightarrow \frac{\partial I}{\partial x} \; for \; x \rightarrow 0$$

$$b_0 = \frac{I(x_0,y_0 + y,t_0) - I(x_0,y_0,t_0)}{(y_0 + y) - y_0} \rightarrow \frac{\partial I}{\partial y} \; for \; y \rightarrow 0$$

$$c_0 = \frac{I(x_0,y_0,t_0 + t) - I(x_0,y_0,t_0)}{(t_0 + t) - t_0} \rightarrow \frac{\partial I}{\partial t} \; for \; t \rightarrow 0$$

Thus, e.g. $(I(1,3,0) - I(2,3,0))/(1-2)$ and $(I(3,3,0) - I(2,3,0))/(3-2)$ are both approximations of the derivative with respect to x at point (2,3,0). To get useful values that are robust to noise we might have to smooth the area or take multiple measurements. Doing so we have a new equation describing the slope of our goal functions at t_0.

5) $$a_0 * v_x + b_0 * v_y = -c_0$$

where $v_x = v_x'(t_0)$ and $v_y = v_y'(t_0)$. This is a function with two unknown numbers (the slopes v_x and v_y). Such an equation cannot be solved either. To solve this we bring in our second assumption: neighboring points move similarly. This means that neighboring points will have the same derivatives at the same t. So we can write the same equation (5) for the neighboring points and thus get multiple equations with the same two unknowns. Instead of having too little information, we now have too much. The equations are over-constrained and probably have no solution that satisfies all the equations. So to solve this we make a best approximation using the least square method.

Estimating position from slope, iterations and scale

Finally we have approximate values for the slope of our goal functions at t_0. What can we use this for? We are not interested in the slope, but in the x- and y-values at time t_1. We can estimate this value using Newton-Rhapson-like iterations. Having the slope for x and y at t_0, we can estimate x and y at t_1 – assuming the goal curve is a straight line within a small time interval. To get

the position of our interest points we simply add the derivatives – more precisely:

$$\begin{pmatrix} v_x(t_1) \\ v_y(t_1) \end{pmatrix} = \begin{pmatrix} v_x(t_0) \\ v_y(t_0) \end{pmatrix} + \begin{pmatrix} v_x'(t_0) \\ v_y'(t_0) \end{pmatrix} (t_1 - t_0)$$

It proves to be useful to iterate over the process with the translated picture. The algorithm typically converges in about 5 iterations (Bradski & Kaehler, 2008).

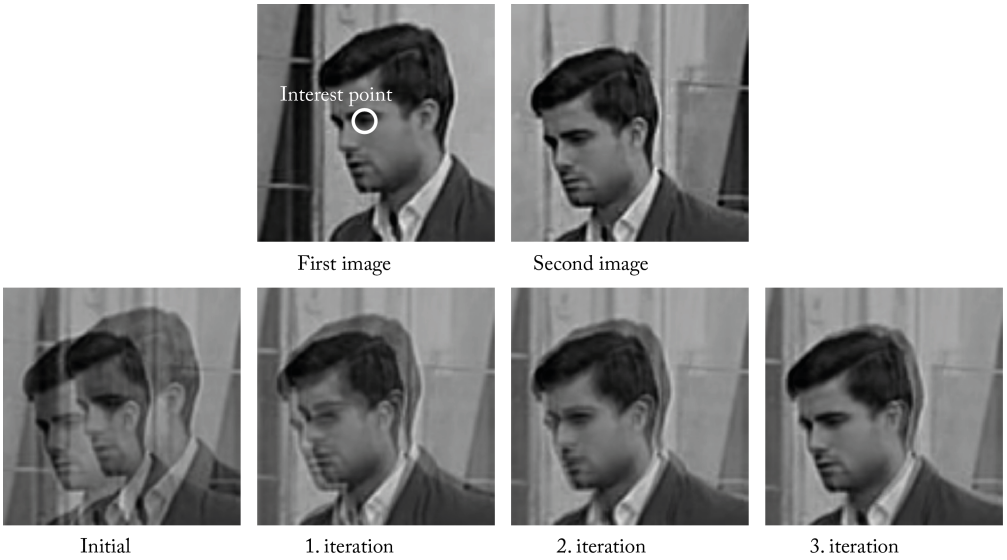


Figure 17 - The initial slope is only an estimate and iterating over the process with the translated picture improves performance.

The third assumption – that the point moves in small intervals only – holds in some situations. But if the object moves fast relative to the frame rate, the algorithm is likely to fail. To counter this we can make our first position estimate using a scaled down version of the images. This will allow us to make a crude translation first and make finer and finer adjustments until we reach the actual pixel level.

Figure 18 - Using the scaled down images we can use D to make a crude estimate which is then refined using C, then B and finally A.

POINT SELECTION

Lucas Kanade tracking allows for estimations of point motion flow. Ideally we select points that are easily recognizable in other images. As such it is the same problem once again, just that instead of objects we try to recognize points. But we can choose the points our selves. A classic approach is to track corners. Intuitively this makes sense. Corners are more distinct than other areas. The middle of a piece of paper is not very distinct. An edge is more distinct, but could look like another part of the edge, whereas a corner is very distinct. Typically there are simply fewer corners in an image than bland areas. Aanæs et al. have shown that fixed scale Harris corner detection (Harris, 1988) is one the most stable of the widely used interest point detectors (Aanæs, 2012).

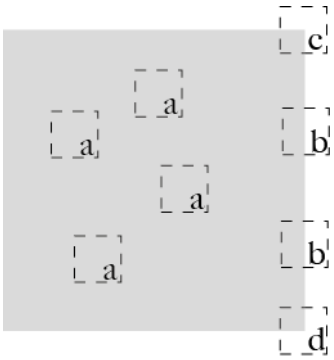


Figure 19 - Corners have a more distinctive look than bland areas and edges. All a's look alike and so does the b's. The corners c and d are unique.

To keep things simple I have simply chosen a grid of points uniformly dispersed over the area to be tracked. This is also in agreement with the original article. This is fast, but makes for more unstable points. The grid density can be increased to ensure more stable tracking at the cost of speed.

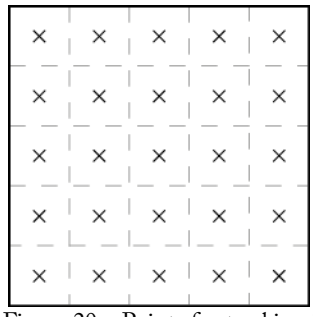


Figure 20 - Points for tracking is uniformly distributed on the bounding box. This is in contrast to other approaches were a more expensive preprocessing step finds good tracking points.

Points are reinitiated at every frame, making the tracker very adaptive. If we kept following the same points, we would loose many points during tracking if some could not be tracked at times due to occlusion of object transformation or other failures. Initializing the points at each frame allows the object to change appearance completely and still be tracked. It also has the less desirable effect of making the tracker prone to drifting. A small drift between frames will easily build up, as the new points might get located on the background and further accelerate the drifting.

MEDIAN FLOW

We now have the tracked points, but what to do with them. We can consider each tracked point as a vector pointing in the direction of its movement. This vector is easily found by subtracting the starting point from the position in is tracked to. But there is no guarantee that they all point in the same direction– in fact they rarely do. They might point in different directions due to tracking errors, scaling of the object or transformation of the object (e.g. during walking where the feet change position relative to the body). So given this cacophony of vectors we need to decide on a single transformation that we believe describes the overall translation of the object.

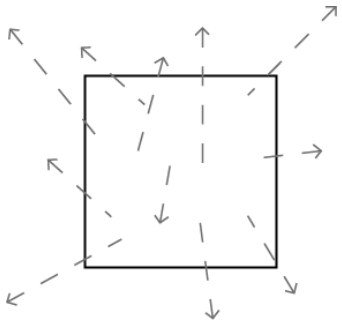


Figure 21 - It is not always trivial how to interpret the flow vectors.

The median flow approach does not take transformations such as changed object ratio, rotation and perspectival transformation into account. Instead the algorithm tried to describe the transformation the best it can with only scale and position. The displacement and scale of the new bounding box is calculated using the median flow algorithm.

Filtering

Not all points can always be tracked, and so the point tracker might discard some points as being too insecure for tracking. The remaining points are tracked with a certain confidence. Instead of using all points for tracking the method prescribes using only the 50% most confident. This is independent of whether they are all very confidently tracked or all poorly tracked. All we know is that it is the most confident. The tracking error can be calculated a number of ways. I use the sum of squares (SSD)(Bouguet, 2000).

Translocation

We know have a bunch of vectors, but how to interpret them as a translation? The translation along the x-axis is then calculated as the median of the vectors x-coordinates and likewise for the y-axis.

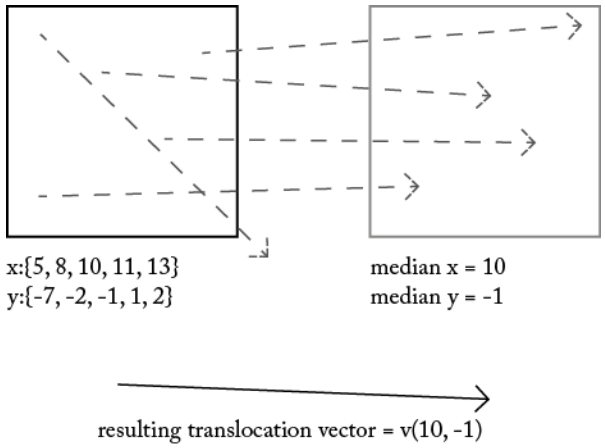


Figure 22 - The movement of the bounding box is calculated from the median flow.

Scaling

To decide how to scale the bounding box, all point-pair combinations are evaluated for their relative difference in distance. Scale is determined as the median of these relative distance measures.

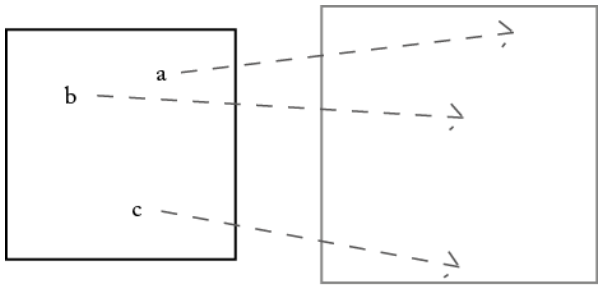


Figure 23 - Scale is determined as the median of relative scale change between all point pairs.

POTENTIAL MODIFICATIONS

The described and used tracker gives us a good idea of the steps involved in tracking and also an idea of parameters which one could experiment with.

- Point selection: Using Harris corner detection might improve performance. The quality of the grid selection is the uniform distribution. Simply choosing the most stable corners do not guarantee this, as the 'best' corners might all be located in a sub region. One way to ensure this could be to use corner detection within grid cells. Another approach, which I will test later on, is to let points float within their cell (Matas & Vojir, 2011). Or we could simply use a finer grid resolution.
- Filtering: Instead of simply relying on the error measure provided by the tracker, which is the SSD, one could expand with other confidence measures. E.g. normalized cross correlation (explained later). Or consistency with previous flow vector under the assumption that points typically flow in the same direction for more than one frame. Or – under the assumption that nearby points on the object move similarly – use a neighborhood consistency constraint, trusting points that move the same direction as their neighbors more than singletons. Detecting regular outliers might also be used to estimate the probability of a point's value (Matas & Vojir, 2011).
- Interpretation: Median flow might not be the optimal way to interpret the transformation. One could imagine using values other than the mean (e.g. a weighted average) or interpret the transformation using Ransac.
- Representation: The current version insists on representing the tracked object as a bounding box orthogonal to the image frame. Allowing perspectival transformations or rotations might increase performance. Or we might use another representation than the box altogether.

CLASSIFIER

Any classifier that use labeled training sets can be used. But if it is to work online it must adapt to new training samples fast and classify even faster. Randomized forests have proven to be both speedy classifiers and fast to train. The implemented classifier is a variation of a sequential randomized forest (Breiman, 2001)(Özuysal, Fua, & Lepetit, 2007)(Kalal, Matas, & Mikolajczyk, Online learning of robust object detectors during unstable tracking, 2009). The structure allows for fast growing, pruning and querying. The forest consists of trees, which in turn consists of features. To supplement the forest a model of examples are also used. Whenever a patch is added to the forest it is also added to the model. If the forest approves of the patch, it is – as a final filter – compared with the model. If it matches the model as well, it is approved as an instance of the class. So to understand the classifier and how it is used, we need to look closer at:

- Features
- Trees
- Forests
- Model matching
- Scanning

FEATURES

An image patch can be described by a number of features. A feature is a simple way of describing either a part of the image of the image as a whole. Examples could be the location of corners, dominant color in a region or location of edges.

I have decided to go use Haar-like features similar to AdaBoost (Viola, 2001). The chosen features are a two-bit pattern. The feature outputs four possible codes that each describes the area with respect to gradient orientation. These are extracted by measuring image intensity: is the upper half darker than the lower half and, is the left half darker than the right half. These two simple binary tests give us four possible outcomes. These features can be extracted very efficiently using integral images.

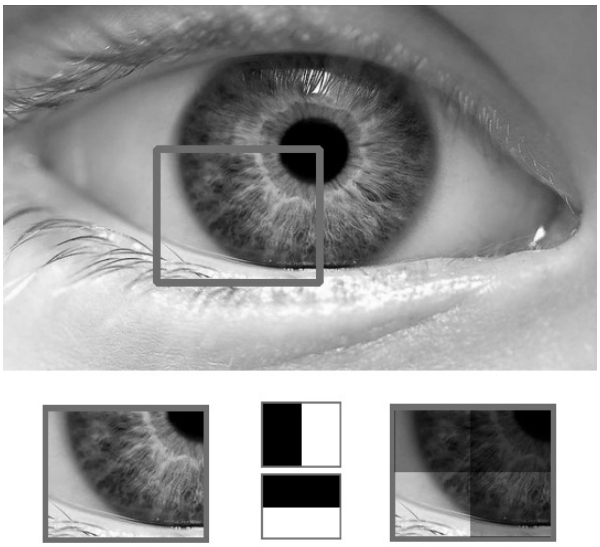


Figure 24 - Features are chosen at random and evaluated using 2 simple binary Haar-like features, giving 4 possible outcomes.

Integral image

An integral image is an image, where each pixel value is the summation of all the pixel values that are above and left of it in the original image. So the upper left pixel will keep its original intensity. The upper right pixel will be the sum of the top row pixels and the bottom right will be the sum of all values. This operation is of course not without cost. But once the integral image is created it allows us to calculate the sum of any sub rectangle, parallel to original image, in constant time.

2	3	1	7
5	4	8	7
9	8	6	2
6	2	5	6

Original image

2	5	6	13
7	14	23	37
16	31	46	62
22	39	59	81

Integral image

Figure 25 - Example of an image and its corresponding integral image.

To get the summed pixel intensities of a rectangular region, we simply add the lower right to the upper left and subtract the two other corners; a total of four lookups and 3 additions/subtractions. When dealing with large areas and when we need the intensity sum of many rectangular regions, the cost of creating the integral image is negligible.

2	3	1	7
5	4	8	7
9	8	6	2
6	2	5	6

Area = 4+8+7+8+6+2+2+5+6 = 48

Rectangle area from original image

2	5	6	13
7	14	23	37
16	31	46	62
22	39	59	81

Area = 81+2-13-22 = 48

Rectangle area from integral image

Figure 26 - Using the integral image we can calculate the sum of intensities in a rectangular area in constant time.

Feature extraction

A feature can have any scale, position and aspect ratio. Wherever it is in the image it describes that region with the aforementioned four codes. An image patch is described using any number of these features – each generated at random and describing a region of the patch.

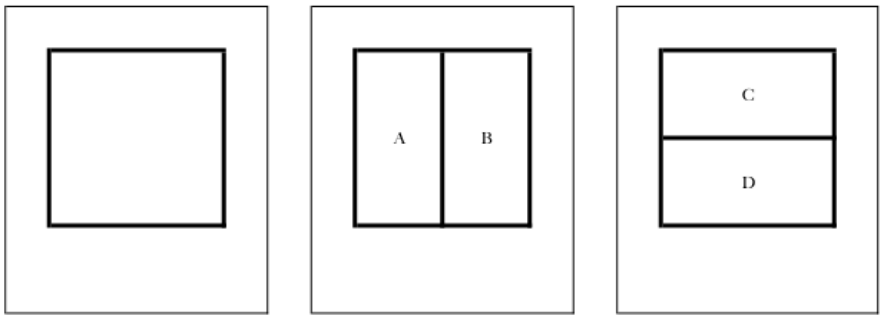


Figure 27 - When having decided on a feature area, the area is divided into four sub areas. "A larger than B" represent first bit of the result and "C larger than D" the second bit. The output of the feature is thus 0, 1, 2 or 3 (binary 00, 01, 10, 11).

2	7	14	17	18	27
3	15	26	34	43	52
5	23	37	53	68	86
12	36	54	75	97	123
13	37	57	81	111	144
14	40	67	96	131	167
23	57	91	129	167	203

Result of first bit is zero if left side, one if rigth side.

A = 57+2-14-13 = 32

B = 111+14-18-57 = 50

Right side is brighter than left side.

Result: 1

2	7	14	17	18	27
3	15	26	34	43	52
5	23	37	53	68	86
12	36	54	75	97	123
13	37	57	81	111	144
14	40	67	96	131	167
23	57	91	129	167	203

Result of second bit is zero if top, one if bottom.

C = 68+2-18-5 = 47

D = 111+5-68-13 = 35

Top is brighter than bottom.

Result: 0

Final result is area BC or binary 10 or integer 2.

Figure 28 - Example of how to calculate the 2-bit feature from an integral image.

Tree

A tree looks like a tree when illustrated and is thus named so. Except it is typically depicted upside down with leaves at the bottom. It has a depth and a branching factor. The depth is the number of times the fern forks and branching how many sub parts a fork splits into.

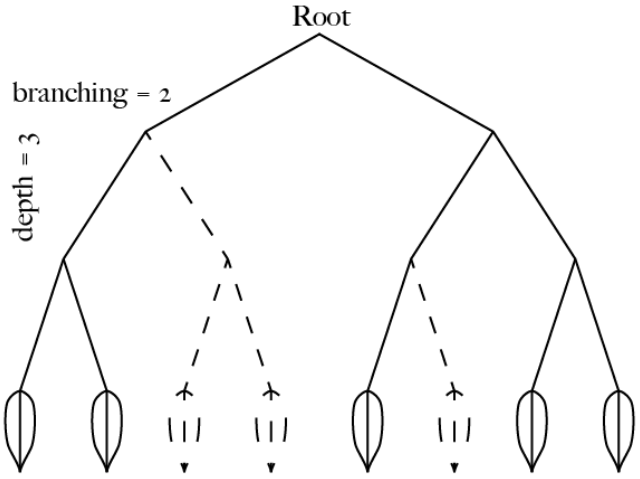


Figure 29 - A fern with branching factor 2 and depth 3. Not all branches are necessarily grown at all times. Branching is in our case a question of how many it can maximum split into. The depth is where we find the leaves.

The tree classifies by asking a question at each junction and storing an answer in each leaf. So if we have a tree that recognizes apples it could start be asking about color. This feature is then extracted from the image patch and we get a result. The result is then used to decide which branch of the tree to explore further – if red, we go down the red branch and so forth. If the branch is missing we consider the sample rejected. If the branch exists, we ask the next question: is it round? The amount of branching depends on the question. If we reach a leaf, we have concluded that the sample is an instance of our class 'apple'. The classifier might not be correct, but that does not change the fact that it is an apple classifier. It is just not a perfect classifier.

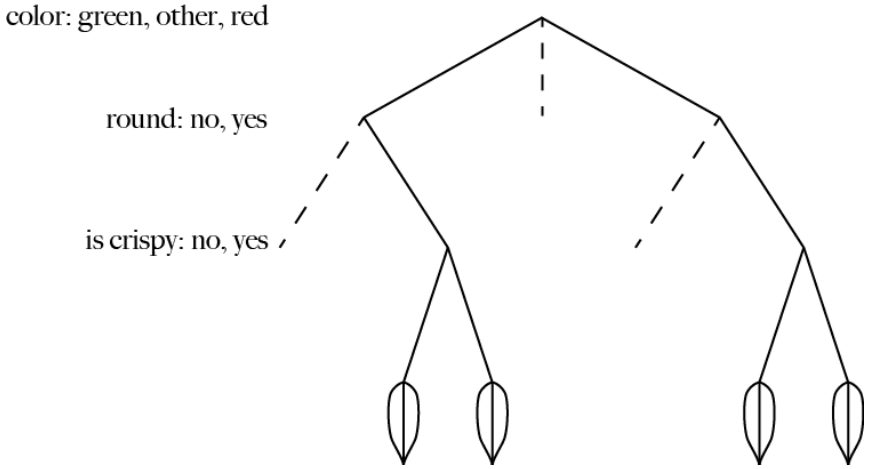


Figure 30 – When querying we evaluate one feature at a time as we move down the tree. If a feature evaluate to a non-existing branch, the patch is discarded.

So at each junction we ask a question. These questions are our features. The depth is the number of features and branching is the possible outcomes of each

feature. Each depth in the tree corresponds to a feature. At each level in the tree a new feature is extracted from the image patch. Each leaf thus represents a possible image patch, as the forest views it. Using these features, a tree using ten features to describe an image patch will be able to represent 4^{10} different patches. The features are how the patch appears to the classifier. If two image patches resolve in the same list of features, the two are considered alike.

Choosing feature areas

We have seen how to extract a feature from the value that gives a low-level description of the area. But how to choose these? Given an image patch we wish to describe this can be done in several ways. One way is to select a rectangle at random within the patch and use this as a feature. So given an image patch we wish to describe with our features, we decide on a number of random sub rectangles, say 10, we then evaluate each rectangle, giving us ten numbers (0,1,2 or 3). These ten numbers gives a crude description of the image and describes how to reach a leaf in our tree. The rectangles are decided upon at creation of the tree, and the same rectangles are used throughout.

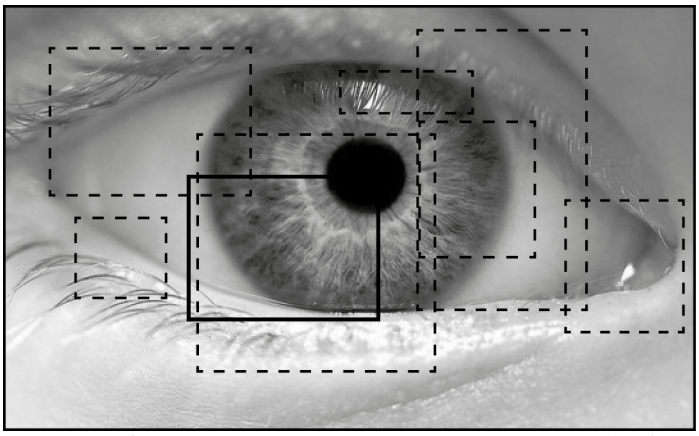


Figure 31 - Given an image patch we want to recognize in the future, we can look at 10 random sub patches and describe these using our features. This will give us a crude model of what the object looks like.

The tree is build using the labeled samples from the training set. A positive labeled sample results in a growing event and negative samples in a pruning event.

Growing

When growing the tree from a sample, all features are extracted, giving us a vector that is created as a branch in our tree. In the beginning the tree is empty, containing only a root node. Then we add our first patch (the one to be tracked). This results in a single branch. Growing is done incrementally, one branch at a time, when we label another patch in our set of frames. If a branch already exists, nothing happens to the tree.

No restructuring of the tree is done at any time. There might be a performance gain in evaluating the most distinctive features first (meaning the one that has the least branches).

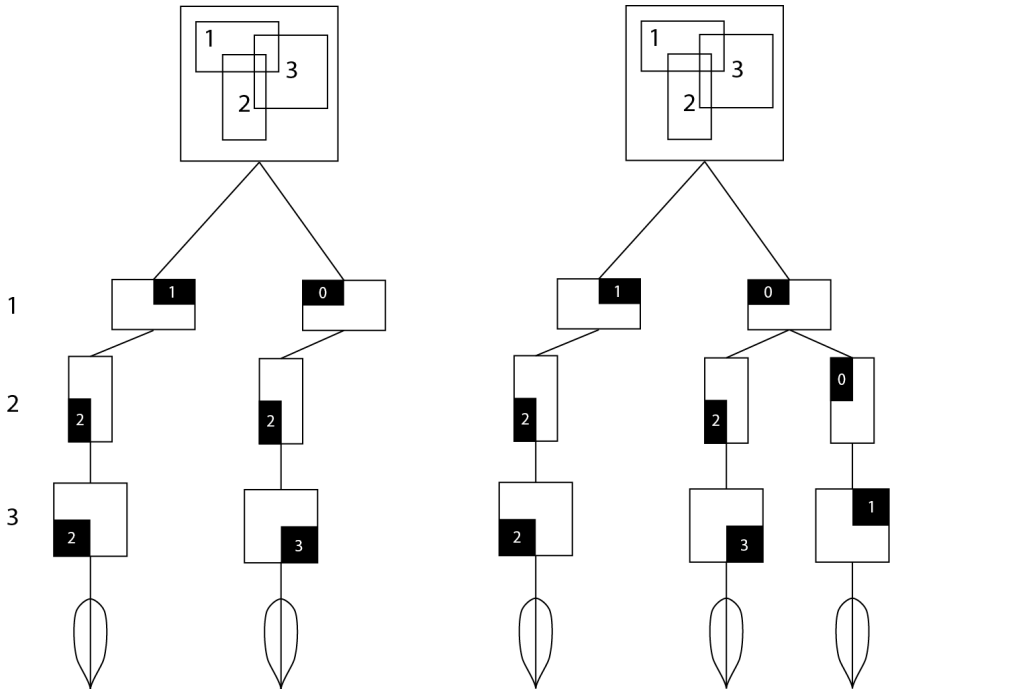


Figure 32 - Left: the tree after to branches have been grown (branch 1,2,2 and 0,2,3). Right: the left tree after an additional growth of a patch where the features evaluated to 0,0,1.

Querying

Querying can be done very efficiently as the structure is naturally ordered as a cascade. We start at the root and evaluate one feature at a time. This means that we rarely will evaluate all features, as most will be discarded in the early stages. If we reach a leaf, the patch is considered a positive example, if not, a negative. The sequential structure makes real-time querying possible. We start out by evaluating feature 1, if the result corresponds to a branch we continue to feature 2. If not the patch is discarded and no more feature need to be evaluated. Most patches will be discarded early on. Whether the first or last feature discards a patch matters not. Only if a leaf is reached does the tree returns a positive answer.

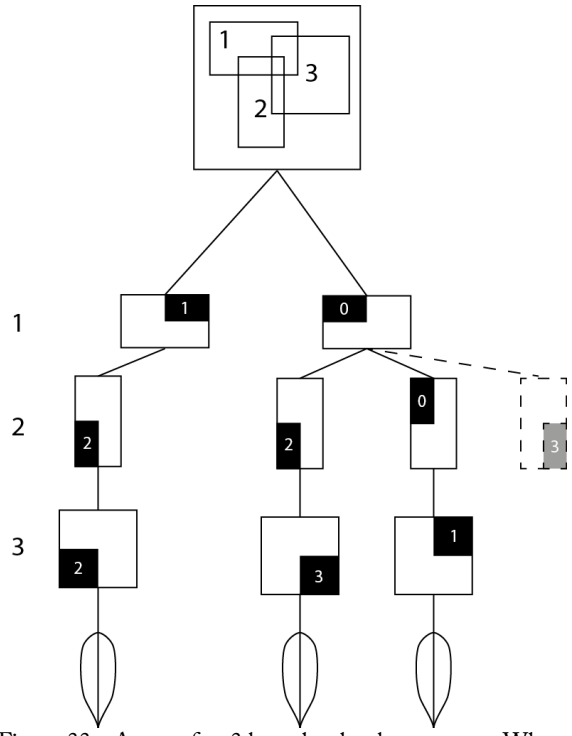


Figure 33 - A tree after 3 branches has been grown. When queried feature 1 is evaluated first, if it is either a top-left or a top-right result, no more features will be evaluated and the tree returns false. If 0 we evaluate feature 2. If feature 2 evaluates to 3 in above example, the tree return false as there is no such branch. This means that only 2/3 of the features were evaluated. If a queried patch reaches a leaf, the tree accepts the patch.

Pruning

Pruning is done when encountering an erroneously labeled sample. When so, the feature prescribed branch is removed. Pruning is done when tracker and detector agrees on object position. In this case all other patches is considered negative. But pruning them all would be unnecessary harsh, as this would prune the forest even when it had not misclassified. Thus, only positively classified patches are pruned. When doing so all trees are pruned, removing the patch completely from the forest.

Another strategy could be to only remove the branch from some of the trees. One problem one could imagine with this pruning and growing strategy is, that the same branch might be grown and pruned over and over because it was inherently insecure. In this case the tree does not capture the uncertainty and the result simply becomes a matter of whether growing or pruning was the last event.

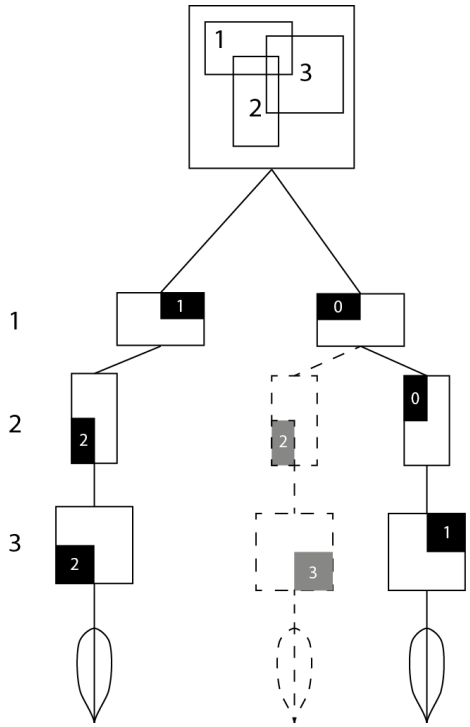


Figure 34 - Pruning of the patch where feature 1=0, feature 2 = 2 and feature 3 = 3. The corresponding branch is removed completely from the tree.

SEQUENTIAL RANDOMIZED FOREST

The classifier is called a forest because it consists of a number of trees. Each tree is a simple classifier of its own. The classifier trees have a binary output: yes or no. The decision of the forest is a majority vote among the trees. A forest can have any number of trees, but since we have to evaluate at least half of them before we can conclude anything, adding more trees will also add processing time. The performance of the forest will increase with more trees but speed drops linearly (Breiman, 2001). As such the forest is a simply umbrella like superstructure, that binds the trees together. The individual trees can be thought of as weak classifiers and the forest a way of empowering them. The idea is, that the individual classifiers do not have to be perfect if we have plenty of them. When one fails, another will hopefully take over. For the trees to supplement each other well they should – obviously – be different and fail at different times. I will look more at this independence assumption during testing.

All trees have the same number of associated features. All features are generated at random, so a forest consisting of ten trees with depth 10 would use 100 features. All trees are grown when the forest grows. When growing a patch all trees are grown. When pruning, all trees are pruned.

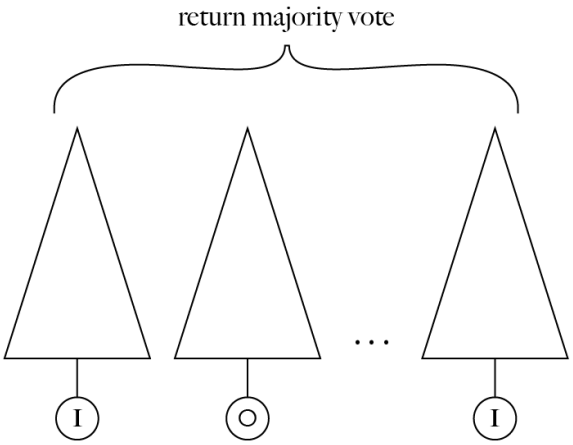


Figure 35 - The forest is a simple umbrella structure binding together a list of weaker classifiers.

MODEL MATCHING

The forest classifier is fast and simple, but as the features are very simple there is also the danger that it might oversimplify the problem and include too many false positives. To avoid this Kalal et al. (2009) suggests using template matching as a final qualifier. This is done by matching the patch against a model using normalized cross correlation (NCC). So every time a patch is added to the forest, a scaled 15 x 15 image is added to the model. And as a final filter, when the forest has approved the patch, the patch is matched against the model. If there is a match with a value above some threshold, the patch is accepted as a positive.

Matching

Matching against the whole model can be expensive if the model is large. And when pruning we need some way of deciding whether to remove the patch or not from the model. Further when querying there is no guarantee that the forest leaves and the found model patch are related. The forest and model might accept the patch for different reasons, which might or might not be a problem. To avoid these issues I have decided to edit the original classifier to only match against those patches that match with the found leaves. As such the forest not only help to filter when to query the model, but also decides which model patches that are candidates for a match. This also makes pruning simple, as pruned branches and corresponding patches, is never matched against.

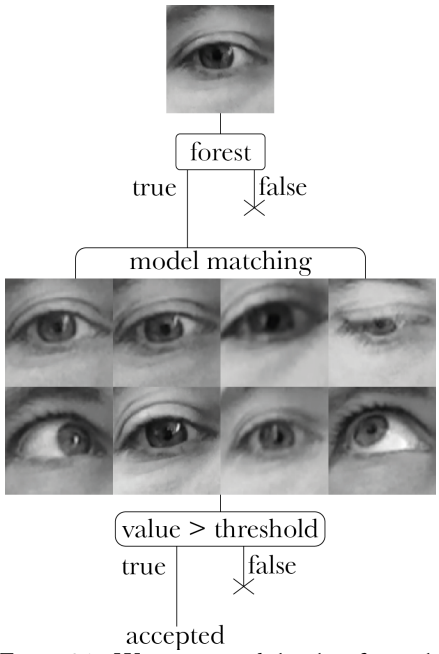


Figure 36 - We can expand the classifier with some additional – and more expensive – classification procedure e.g. template matching. The forest will then work as a fast filter ensuring that only the most promising candidates are analyzed further.

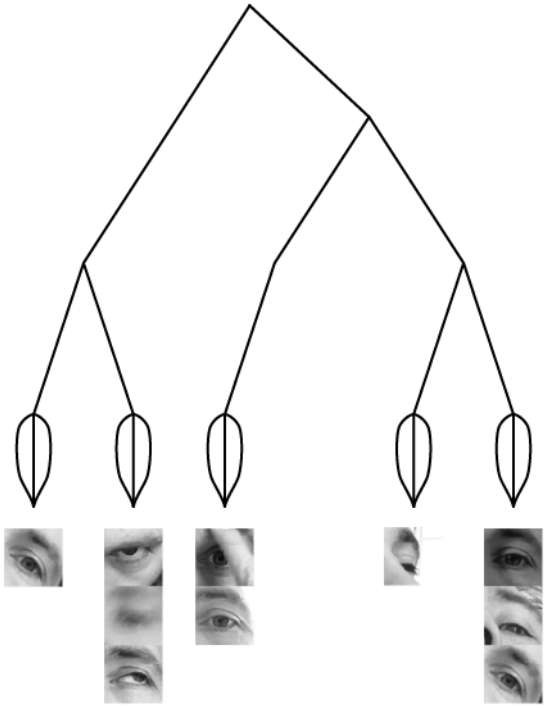


Figure 37 - Storing corresponding patches at leaf nodes, for comparison as a final filter using NCC.

SCANNING

When searching for an object in the image a scanning approach is used. The idea is to slide a search-window across the image. Ideally we look at all possible locations in all possible scales. In reality we do so in intervals, hoping that our classifier will identify the object even though the window is not at the exact position and have the exact size. If the object has alternating aspect ratio, we might need to scan with different ratios as well. For our purpose, scanning with the original aspect ratio will have to suffice. For each window we ask the classifier whether this sub patch of the image is an instance of our class. If multiple windows are true, we will need to handle this. As our algorithm assumes that the object is unique, we will have to choose the most likely. To do this the patch with the highest model similarity is chosen.

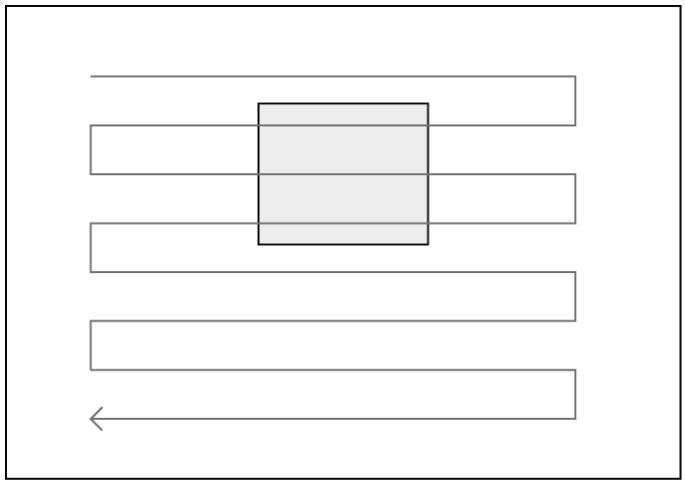


Figure 38 - A sliding window is used to scan the image for the object. Several sizes of the window are used to ensure that we find the object whether it is small or large.

I have chosen to move the window in intervals of 10% of the window size and increase size by 25% starting at 70 pixels (least of the dimensions). Using a 640 x 480 pixel image, this amounts to roughly 4500 image patches. The choice is a compromise between speed and quality. A faster computer or implementation might allow for a higher resolution. Kalal et al. scans at 10.000 positions. One could also consider doing a crude scan first, and then do local optimization in areas that seems promising. Each window patch is evaluated using the classifier. First step in the classifier is the forest.

PARAMETERS

There are several parameters that can be adjusted in the classifier. We can of course, use different features altogether, but there is also the random selection of these within the patch. When nothing else is mentioned, the feature

rectangle will be generated using a uniform random generator: position and size. Then there is the minimum size of the features. When nothing else is mentioned, an arbitrary lower threshold of 20% of minimum object dimension is used. So an object whose height is twice the width will have a minimum feature size of 20% of the width at any given time. Further we can preprocess the image before analysis (e.g. smoothing and equalization). As a standard no such preprocessing is used. Then there is the amount of trees and the depth. As a standard, 9 trees are used and they have a depth of 10. Further we can grow with different strategies (e.g. adding affine transformations as well). As a standard only the patch in its simplest form is added. In tests I have been interested in the performance of the forest and has therefore not used template matching as a final qualifier unless explicitly stated. This has been done to understand the forest separately and to see how well it performed as such. When evaluating features a threshold can be used to ensure a minimum intensity difference. This has not been done. So even if the intensity difference is a million versus a million and one, this will be interpreted as a valid difference. Pruning is done to sanitize the model from erroneous added patches; this is not used during testing unless specified, as we know exactly what have been added.

SUMMARY OF METHOD

The TLD long-term tracker combines tracking by optic flow with tracking by detection, while at the same time building a classifier. The idea is to use the tracker to collect training samples to train a classifier that in return can be used to track by detection when tracking by optic flow fails. Patches on the tracked trajectory are added to the classifier using a loop strategy. The loop strategy means that if the classifier recognizes two patches on the trajectory, then intermediate patches are added. This allows for adding a wide range of object appearances to the model.

For short-term tracking the Median flow tracker has been used. The tracker is based on point flow tracking using the Lucas Kanade method. No advanced point selection strategy is employed but simply intersections from a rectangular grid. These points are then tracked using LK. The 50% most confident points are used for estimating position and scale. Position is calculated as the median flow. Scale is the median of the relative scale changes between all point pairs. New points for tracking are selected in each frame. This makes the tracker very adaptive and very prone to drifting. Drifting is handled by initialization using the classifier. The adaptive nature makes it possible to add various appearance transformations to the model.

The classifier is a version of a randomized sequential forest. The forest consists of trees. Each tree is in itself a weak classifier. The decision of the forest is a majority vote of the trees. The trees are constructed from features. The features are simple patterns with 4 possible outcomes calculated from image intensities, describing the grayscale gradient. An image patch is described by a number of features with random position, ratio and scale. These features are randomly portioned into groups, corresponding to each tree. Each group, or tree, represents a different way of viewing the patch. The response of the group is a vector and thought of as a branch on the tree. When adding a positive patch a branch is added. When adding a negative patch, a branch is removed from the tree. For a tree to accept a patch, the patch must be identical to an existing branch. The features are evaluated one at a time. This cascading nature of the trees allows for fast evaluation of the patch. As a final filter, when the forest as a whole has accepted the patch, a NCC match is performed to ensure that the patch looks like a template in the model.

TESTS

The TLD framework, tracker and classifier have now been introduced. But we have seen neither in action and thus have little idea of how they handle the previously described challenges. In this section, the tracker, classifier and finally the framework will be tested.

TESTS

To have near perfect conditions for tests I have decided to work partly with synthetic data. Tracker and classifier will mainly be tested on synthetic material and the TLD framework on actual video recordings. Tracking will be tested first, followed by the classifier. This will give us an idea of their workings and weaknesses before we combine and test them in the TLD framework.

TRACKING EXPERIMENTS

INTRODUCTION

Test results are summarized as a single number describing the minimum coverage through the video or alternatively as coverage per frame. Coverage is here defined as the minimum value of precision and recall. If A is the hypothesis, B the ground truth and C the union, precision P is defined as C/A and recall as C/B. Coverage with respect to frame is the minimum of the two. Coverage in a sequence is the lowest in the sequence.

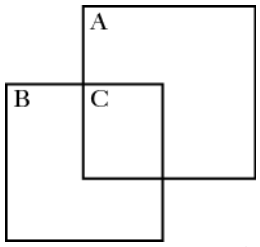


Figure 39 - Precision is defined as union (C) over hypothesis (A). And recall as C over ground truth (B). Coverage is the least of the two.

The tracker experiments have been divided into three sections:

- Which objects.
- Which circumstances.
- Which transformations.

The first step is to get an idea of which objects can be tracked. The second to understand under which circumstances the tracker works. And lastly which object transformations that can be handled.

WHICH OBJECTS

A natural question to ask is which objects we can track given the proposed tracker. I will in these tests consider texture, cropping and object form. I will start with some extreme synthetic cases and afterwards look at real world videos.

Synthetic surfaces

We are here interested in the surface and its relation to the background. That is, which textures can we track and does the background matter? These tests will give us an idea of what factors are in play and give us a chance to relate the results to what we can expect given what we know about the trackers composition.

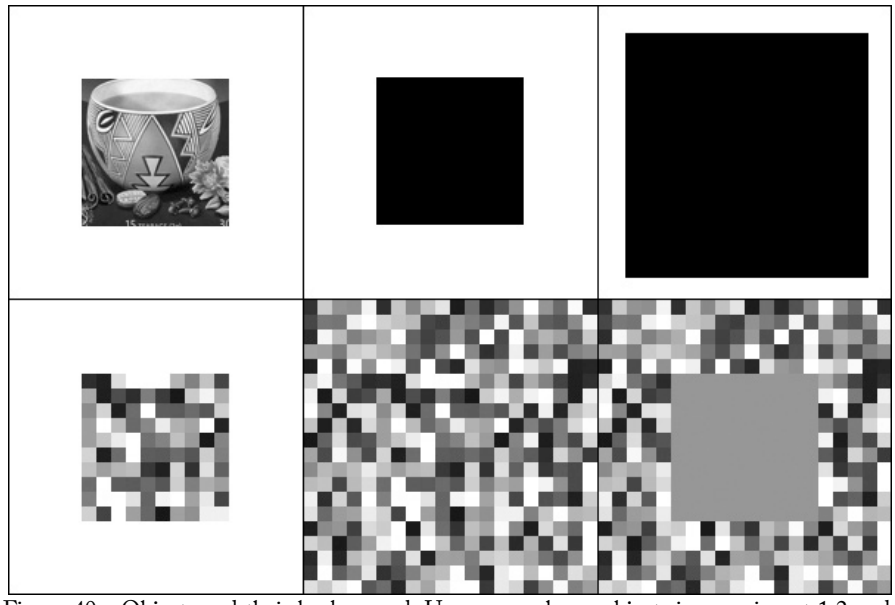


Figure 40 - Objects and their background. Upper row shows objects in experiment 1,2 and 3. Lower row experiment 4,5 and 6. Noise has been enlarged in example 4,5 and 6 to be viewable in print. Same area is tracked in all cases (so in 3 only the center of the black box is attempted tracked). Object in 5 is the same as in 4, but impossible to distinguish from background when not moving. See videos 01-06.

Setup: We will look at six scenarios. In all cases the object moves slowly across the image with steps of 0.1 of tracked patch width. In all cases but 3) is the tracked patch equal to the object.

1) Textured object on uniform background: A picture with high degree of texture and pattern on an all white background.
2) Black object on uniform background: The picture is replaced with a black square.
3) Large black object on uniform background: A larger black object is inserted. Same area as before is tracked, but since the object is larger we only track the middle area of the object.
4) Noisy object on uniform background: The same object size as in 1,2,3. Object is ultra textured being made of white Gaussian noise (same in all frames).
5) Noisy object on noisy background: Same object as in 4, but background is replaced with similar noise.
6) Gray object on noisy background: Same background as in 5, but object replaced with uniform object with same average intensity as background.

Results are simply marked as success or failure. This is done since in all cases, the tracker either tracked perfectly through the entire sequence or lost track in the first frame.

Results: All objects moved in the same pattern, and yes some objects are perfectly tracked while others fail immediately. I will discuss the cases individually.

Scenario	1	2	3	4	5	6
Result	Success	Success	Failure	Success	Failure	Failure

1) A simple case and luckily perfectly tracked. The object has a lot of pattern and is very different from the background. If we look at the point flow we see that all points are tracked nicely in all frames.
2) This object is also tracked perfectly. It has absolutely no pattern, but is distinct from the background. If we look at the points flow, we see something very different from previous object. Only the two corners points are tracked. This makes sense given what we know about the tracker. Since there is no difference in image intensity between the two images at most of the tracked points, there is nothing to track. These points look like all other points in the area so there is no gradient. The only distinct areas are, as discussed earlier, the corners. And only two are tracked, since we only use the most confidently tracked half of the points.
3) Complete tracking failure. None of the points can be tracked since there is absolutely no texture and not even the background contrast can be used.
4) Successfully tracked. These seem to be no such thing as too much texture[2].
5) Tracking failure. The tracker is stuck at the initial position. This is obviously due to the change in background, since everything else is as in 4. Looking at the flow vectors, we see that all points are tracked with great uncertainty and further judged not to have moved. What has happened is that all points look alike. Indeed it is a difficult task for the human eye as well. One way to solve this is by minimizing the LK window, but this is not a universal solution.
6) Tracking failure. Unlike 5 the human eye easily tracks this sequence. Looking at the point flow we see the same as in 5, but unlike before adjusting the window size is of no use. Using a small window the entire object looks the same (being uniform). And if the window is enlarged to

[2] This of course is dependent on the LK window size. "There is therefore a natural tradeoff between local accuracy and robustness when choosing the integration window size" (Bouguet, 2000). So in theory we would like our window fairly big to allow for internal tranformations, noise and ensure a unique area, but not so large that background is included or the local accuracy lost.

include the unique corners, the average value becomes the same. In essence no points and no window size solves the problem.

Discussion: As we can see from the results, texture, cropping and background makes a huge difference. We have learned that the tracker prefers patterned objects. Trying to track uniform objects we are left with the areas near corners and if no such corners are near our tracked patch, the tracker fails. We have further seen the influence of the background. Tracking an object on a background that has similar intensity values can be a problem. And there even exists scenarios where tracking is not possible using LK tracking, even if we customize parameters.

These experiments also tell us something about how the LK tracker is different from the human visual system. Using a small window the LK tracker can easily track the object in 5. This is very difficult to the human eye. And vice versa, the LK tracker cannot track the object in 6 though it is a very simple task for the human eye. Task 3 also shows us, that tracking a patch solely on the information within the patch is not always a possibility. In cases like these a tracker such as median flow fails. This might be solved by expanding with the earlier mentioned supporters (Grabner, Matas, Gool, & Cattin, 2010) instead of considering the points as objectless entities during tracking.

Surface markings

As previous test made clear, surface markings is important. But it might not be a large issue in many real world scenarios as there typically is some texture. But this is based on synthetic data; it is time to test on real world video sequences.

Figure 41 - 8 objects. First 3 objects have many surface markings. Last 5 has very few. See videos 07-22.

Setup: Eight objects are chosen based on their amount of pattern: three with large amounts of pattern and five with very little. The objects are in all videos kept near still in the middle of the frame and moved across the same

background in approximately the same speed. Two patch selections are tested on each video. One where only a sub patch is tracked and one where the whole object along with some of the background is attempted tracked. The reason for tracking both a) sub patch and b) whole object and some background, is that we in previous experiment saw that points near corners might stabilize tracking when there was no surface markings.

Amount of pattern	Object	Sub patch	Whole object
High	Tea	0.93	0.97
	Matryoshka	0.95	0.94
	Box	0.99	0.99
Low	Apple	0.74	0.87
	Orange	0.17	0.00
	Wood	0.00	0.87
	Cardboard	0.00	0.00
	Napkin	0.00	0.47

Table 1 - Minimum coverage in the tracked sequences. Objects with many surface markings are clearly easier to track.

Results: We clearly see that the highly patterned objects are tracked to near perfection. And the differences there could well be imprecision in the manually labeled ground truth. And likewise we see the tracker fail utterly on the pattern poor objects. Those with a medium amount (as the apple) do mediocre. This is what we could expect given what we have learned in the first experiment.

Interestingly we also see that there is no simple answer to whether including background is helpful. In some cases it seems to be and in others not. As we have already seen, the areas near edges and corners are useful when the object lacks surface markings. But these points only have value if the background is stable across frames. Also points outside the object works as noise. In cases where these points are those that are most easily identified, the tracker fails. This puts a cap on the allowed dispersion of the object form, as we shall see in next test.

Density

Before turning to the tracking circumstances, I will look at some tricky objects with irregular shapes and material.

Figure 42 - the three objects. Pliers are tracked twice with two bounding boxes. The black box is the initial bounding box. White points are the most confidently tracked points. Lines mark the point flow.

Setup: Three objects are moved across same background as in above experiment with surface markings. The first object is a pair of pliers, the second a hand trainer and third a glass container. The pliers are tracked twice with different bounding boxes to see if a closer crop makes a difference.

Results: All 4 videos fail blatantly and this makes sense. The glass container, have no real texture but irregular reflections and thus nothing – or at least very little stable - to lock on to. And as the algorithm consider all points within the initial bounding box equal, tracking dispersed objects are very difficult. Success is then completely dependent on the points on the object being easier to track than the background and on the ratio of object- versus background points. The algorithm breaks when the majority of confidently tracked points belong to the background. Since we decide location by median, the breaking point is when 50 percent or more of the confidence points are located on the background. But in many scenarios this threshold might well be lower. This is the case if points on the background are easier to track. Then the threshold will drop even further. And just as important, if the object is heavily dispersed, then all points, even if they are located on the object, include some measure of the background (due to the window size), and will therefore be influenced by background noise. This makes heavily dispersed and transparent objects near impossible to track with the current approach.

WHICH CIRCUMSTANCES

We have seen that the tracker prefers patches with a rich pattern and that patches ideally only contain the object of interest. But in cases where there are very few surface markings, including some potion of background might actually help to stabilize the performance. We will now turn to the circumstances in which the object is tracked. How does the tracker handle exposure, lighting, shadows and occlusion?

Occlusion

Handling occlusion is essential when tracking. Total or at least partial occlusion is in many scenarios very likely to happen. Occlusion exists in many forms: partial occlusion by another object (e.g. hand covering part of face), total occlusion, self-occlusion (e.g. rotation), out of view occlusion and scattered occlusion (e.g. canopy). And the occlusion can either be due to change in viewing angle or object movement (of either the occluded or occluding object).

Setup: I will test for 4 forms of partial occlusion.

A) The object slowly moves to occlusion. Retracts in opposite directions.
B) The object slowly moves to occlusion. While partly occluded the object moves.

Both are done with occlusion by object and by sliding out of the frame. The sequences are 4 different synthetic objects on different backgrounds. The object moves with steps of 0.1 patch width.

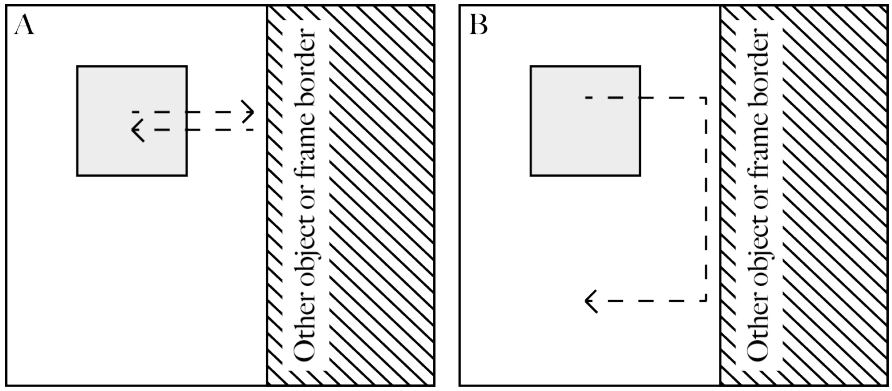


Figure 43 – A: the object is moved to occlusion and back. B: the object is moved while partly occluded.


Figure 44 - Example of object occlusion by other object. Object movement as in B (figure 45). We see that all the tracked points are placed on the occluding object and the background.

Results: Only average results are noted. Standard deviation is in all cases below 0.08. There is a large performance difference between occlusion by frame and object. Occlusion by other object cannot be handled, while there is no problem handling occlusion by frame. As another object increasingly occludes the tracked object, the tracker will at some points stop tracking and instead lock on to the occluding object. This happens when enough points are located on

the occluding object. The tracker is adaptive and initializes points at every frame and this makes it sensitive to occlusion. In A, the tracked object retracts in the same directions as it came from, and the tracker occasionally gets dragged along with the object again (with some lose to precision). One half recouped well, and the other half continued tracking the occluding object. If the object is hidden from view by the frame border and not another object, it performs perfectly. This is because the tracking points does not lock on to another object, but are simply not used. So the only points used will be on the object. There is less data to work with, but all of it is relevant. In case of total disappearance the track is – of course – lost completely, and can only be regained by coincidence.

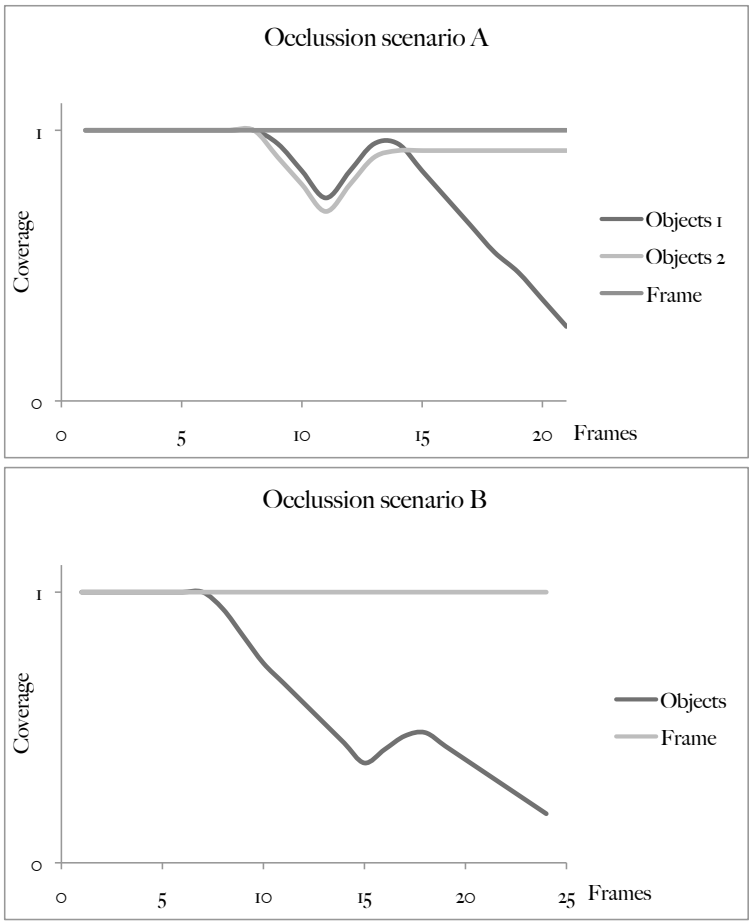


Figure 45 - Tracking coverage over time. In A there are two curves describing occlusion by object. This is because these tests clearly divided into two groups. Those that recovered from occlusion and those that did not. Occlusion begins at frame 4, and at frame 8 50% is occluded. Occlusion by frame border is of no problem. See video 23 for example of tracking failure.

That the tracker looses track are in some sense not a problem if our TLD framework works. When the object is not occluded anymore, the classifier will set in and initialize the tracker anew. But as test A also showed us, the tracker might – successfully – track across minor occlusion. This means that we will risk adding occluded patches to our tree and model. Whether this is a problem is dependent on usage, but it could well be a problem. This is especially

problematic if the patch is very occluded at some points during the tracking; we then risk growing and subsequently initializing on the occluding object instead.

Illumination direction

The intensity appearance of an object can change immensely with illumination. For the TLD long-term tracker to work, we need to be able to track across these illumination transformations. If we cannot, these patches will never be added to the model.

Setup: Using a 3D model of a sphere six scenarios are tested. The sphere is still and a light source is moved across it. The only thing that changes is the texture of the sphere and the background. The sphere is either uniformly colored or a globe. The background is uniform, a still picture or a moving background; all in all 6 combinations.

Results: The results clearly show that both background and object pattern has a great influence on the performance. We once again see that objects with rich surface markings are far easier to track. Tracking a static object on a static textured background is obviously the easiest. The background then simply works as a stable point of reference, stabilizing the tracker. When it is uniform, there is little information to help the tracker. And when the background is moving, it is actually a distraction.

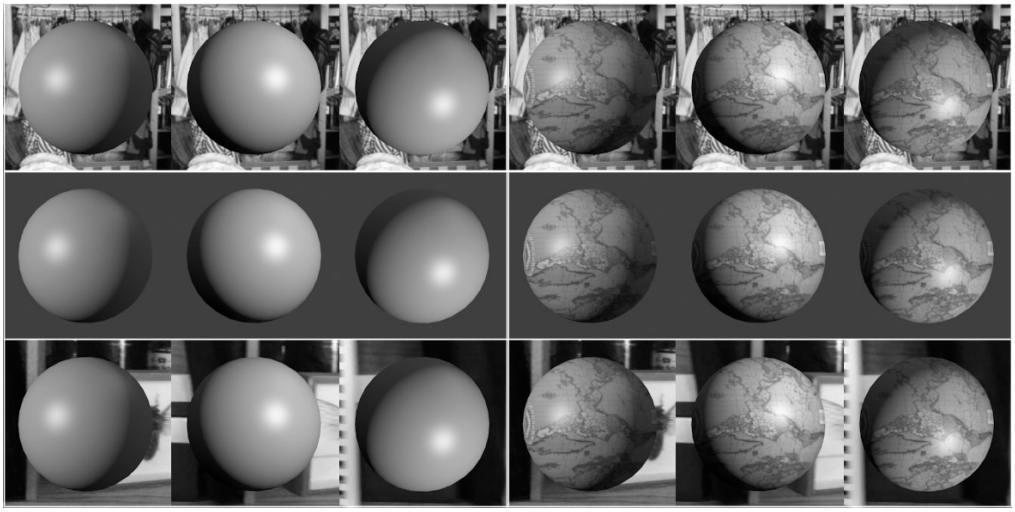


Figure 46 - The 6 videos. 3 frames are shown from each video (first, middle and last). First row has a still image as background. Second row a uniform background. And bottom row a video background.

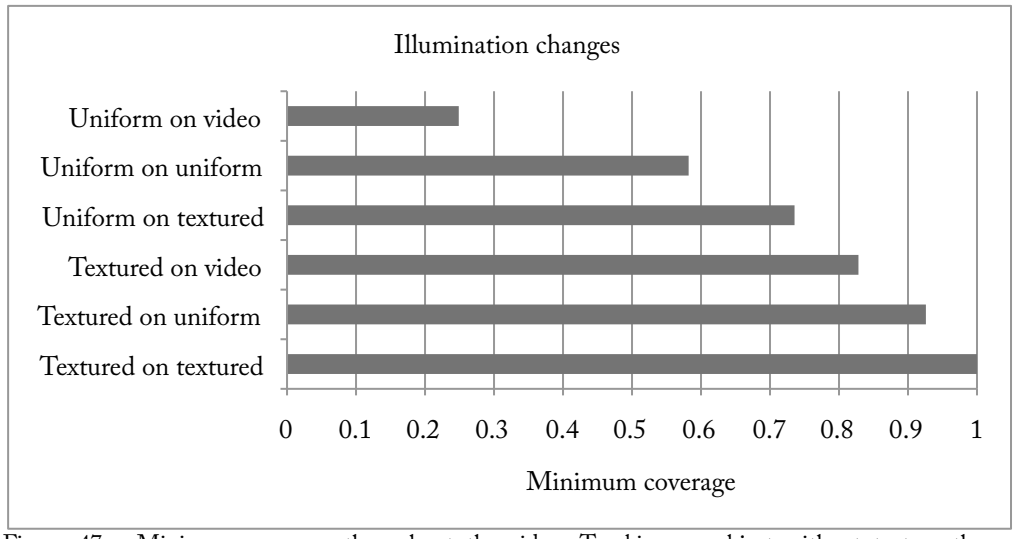


Figure 47 - Minimum coverage throughout the video. Tracking an object without texture through illumination changes lead to tracking failure – especially if the background is changing too. See video 24 for textured on texture and 25 for uniform on video.

Errors happen because the tracker considers the highlight a part of the object pattern and follows the highlight, moving first right, and the downward. In case of the uniform object on the moving background, the tracker first follows the highlight including more background, which then drags the tracker along to the right. In some sense it is only fair that it cannot track the uniform sphere. There really is no telling that it is a light source change rather than a turning object with a color gradient. But to the human eye it seems a more likely hypothesis to assume light change than object rotation – especially so in cases where the background is still. From these synthetic tests it seems like our tracker handles lighting changes well as long as the object is richly patterned. Before we proceed this will be tested on an actual video recording.

Setup: A matryoshka doll with many surface markings is placed on a table and recorded with a near stable camera. The direction of light changes drastically throughout the video.

Figure 48 - frames from video 26.

Figure 49 - five examples of the tracked patch. The tracker succeeds in tracking the patch through dramatic changes in illumination. See video 26.

Results: The tracker performs well on this challenging sequence. Coverage never falls below 0.83. If we look at the tracking frame-by-frame we see that there is one specific place where the tracker jumps of target and performance drops to the 0.83. Between these two frames there is a sudden change in exposure time. This leads me to conclude that though lighting can be a problem for the tracker, it can handle it well if the object is patterned and preferably on a static background. Exposure and more sudden light changes seems a more pressing problem. This will be tested next.

Sudden intensity changes

Previous experiment as well as theory tells us that the tracker does not handle sudden intensity changes well since intensity is what is tracked. This could be a real problem in many real world scenarios. These sudden changes in light can be caused by many factors:

- Automatic exposure correction as a new object enters the cameras focus area.
- A cloud blocks the sun.
- A lamp is turned on.
- Reflections. Such as a blank surface changing angle to the light source.
- The tracked object moves to an area with backlight.
- Shadows.

Setup: To see how large this problem really is, three simple tests are made.

a. A synthetic test. A patch on an image is tracked across two images. The second image is artificially darkened to simulate exposure change.
b. Recording of an orange. Exposure changes throughout video as the camera changes focus from orange to wall and back again (see video 27).
c. Recording of a book with a blank surface. The book changes position in relation to the light source, which creates sudden brightness changes.

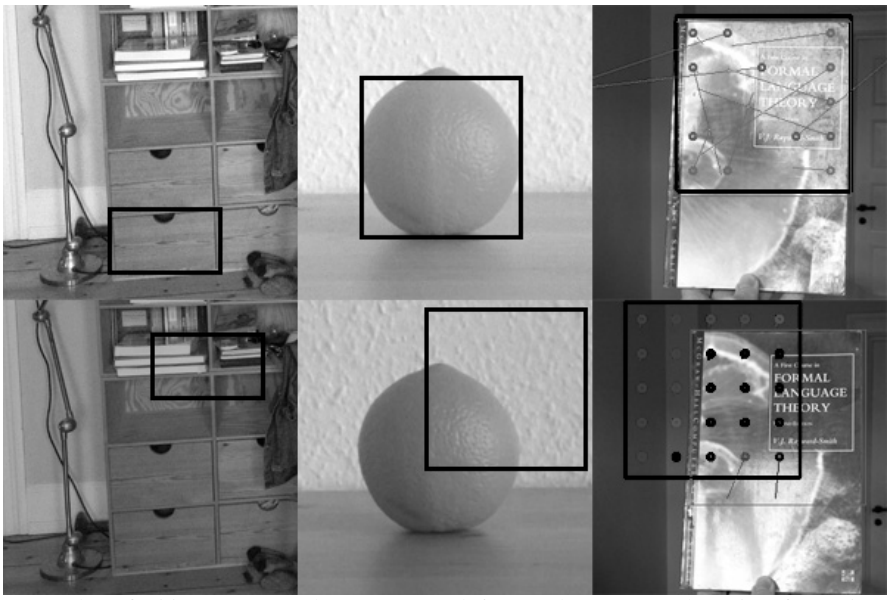


Figure 50 - Sudden intensity changes. From left to right is test a, b and c. See video 27 for orange example.

Results: The results are unambiguous. Whenever exposure changes, the tracker jumps to some seemingly random area. If we look at the tracked points we also see that they point in all directions whenever there is large changes in intensity. This makes sense since the tracker only considers gray levels and not more complex features. This means that exposure changes and reflections can confuse it immensely. Likewise shadows can also confuse it if they change too fast. If we have already added the patch to our model, the tracker might be initialized at the correct position if the only change is exposure. But this requires the patch to be recognized by the classifier and this is not always the case. The better the tracker the better the classifier will get as more correct patches will be added.

WHICH TRANFORMATIONS

We have now looked at which objects are best tracked and under which circumstances. We have yet to see which transformations can be tracked and how well.

Scaling

So far we have only been looking at one kind of transformation, namely translocation. It is time to look at scaling. Before we turn to the experiments I will consider the algorithm. If the used points are tracked perfectly, translocation in itself is always correct and so is the scale change. But the

median flows combination of the two is not perfect. We can see this by considering a simple hypothetical example.

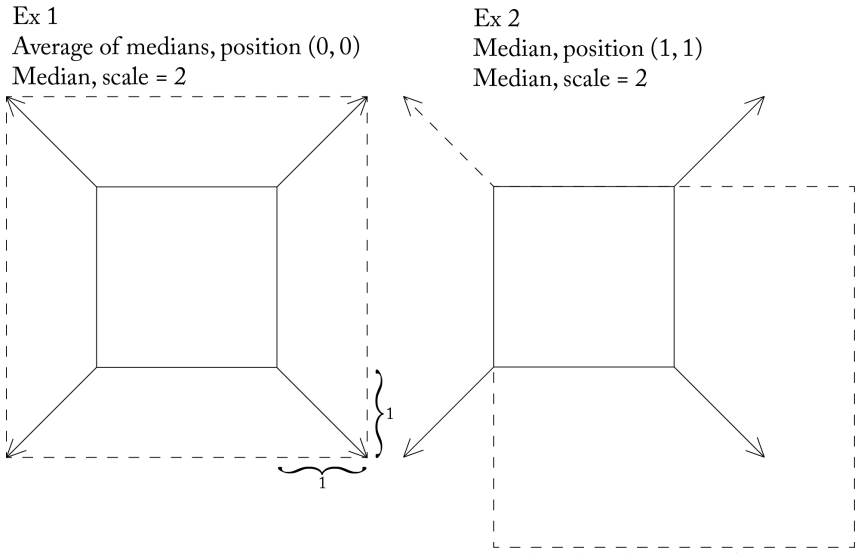


Figure 51 - In example 1 all 4 corners are tracked and used leading to perfect tracking. In example 2, all but the upper left corner is tracked. Even though the 3 other points are tracked correctly, the tracker fails. This is the case when the used points are not evenly distributed.

Consider a patch in where only 6 points are tracked confidently and the 3 above the median is placed in the three corners. If the patch is enlarged but otherwise stands still, the tracker fails. This happens because scaling and point translocation is not independent, so calculating them independently might lead to compensating for the same point flow twice. This observation can be confirmed by a small synthetic test where a patch is enlarged (see video 28). The tracker slides slightly off due to an initial uneven distribution. And once the tracker starts to slide of center, the tendency is reinforced. In the next frame, where the target is enlarged further, there will be more points that flow left, as the tracker has now moved slightly left on the object. This will escalate the error further. Looking at the scaling and location separately they work well if the LK tracker delivers perfectly tracked points. But, when combining the two things go wrong when there isn't uniform point distribution.

Rotation in plane

As with scaling we can say something about rotation based on what we know about the algorithm. The tracker is obviously not designed to track objects that are oblong and rotates as no rotation of the bounding box is allowed. So taking an oblong object and rotating it in the image plane, we would either:

1) Force the tracker to scale to include the whole object along with a good deal of background.
2) Keep current scale and include some background and discard some object or

3) Scale to include only a small portion of the object.

Neither solution seems satisfying. So the tracker is made to track objects that do not rotate in the image plane. But if the object has equal width and height, it is doable, but it isn't build for it.

Figure 52 – Mobile phone on a table. Example of an object that cannot be tracked without having to take some of the above-described unfavorable decision.

Setup: As with scaling point distribution is also an issue when rotating. We can see this by tracking three different objects: 1) a highly symmetric artificial object. 2) A highly asymmetric object. And 3) the same image that lead to failure when testing for scaling. They are all rotated 90 degrees clockwise on a uniform background. Center is in all cases held still.

Figure 53 - A symmetric, highly asymmetric and a not manipulated photo is rotated 90 degrees on a uniform background. The resulting coverage is 0.96, 0.51 and 0.88 respectively. See video 29-31.

Results: The symmetric object is tracked to near perfection with a resulting coverage of 0.96. The asymmetric has a coverage of 0.51 and the picture of the matryoshka 0.88 coverage. Again the drift is due to the fact that the points considered are not evenly dispersed. When a majority of the points are recognized on the left side of the object, the tracker will drift upward if the objects turn clockwise. So if we want to track objects that rotate in the image plane, they should be symmetric in both form and pattern.

Rotation out of plane I

Ideally our tracker is able to handle out of plane rotations such as a head turning from en face to profile. Before testing it is worth pondering what we mean by tracking through rotation. If we imagine a globe where we can see Europe. What should the tracker do when the globe rotates 90 degrees left, turning Asia towards us? 1) One scenario would be that it tracked the actual surface area on the globe with all its deformations, bending et cetera. This

version is not possible for the current tracker that insists on a rectangular bounding box with static dimensions. 2) Another scenario would be that it simply kept locked at the position at the globe as object and thus moved focus to Asia, keeping the tracker in the middle of the globe. 3) A third scenario could be that it gave a best estimate of the bounding box of where Europe is, balancing between including to little and too much as the proportion changed, meaning that the tracker would be centered at the left edge of the globe. The median flow algorithm is essentially a tracker of the last type – a surface tracker. But since it at the same time insists on a rigid bounding box, it needs to make compromises and will at times track the object and other parts of the surface instead.

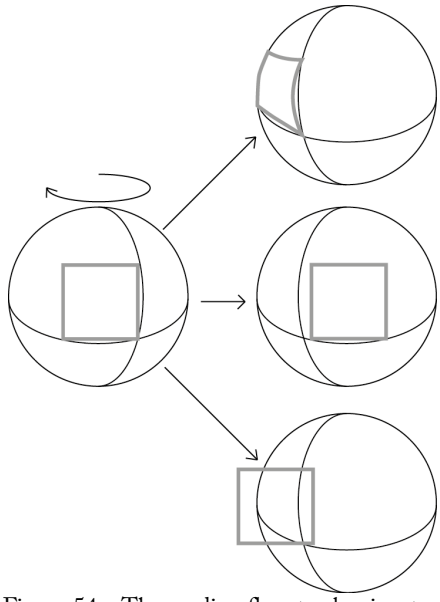


Figure 54 - The median flow tracker is a tracker of the third type. It attempts to update a bounding box, with the initial proportions for the initially chosen surface area.

When confronted with non-linear points compression the tracker strikes a balance between inclusion and exclusion, this can lead to problems, as we shall see in the following experiment.

Setup: To understand what happens during out-of-plane rotations I will look at 4 scenarios involving a synthetic globe: 2 transformations on 2 backgrounds. The transformations are rotation and movement. When rotating the points first float left and then right. When moving, the points move to the right.

Transformation	Rotate 70 degrees and back (A)	Rotate 70 degrees and back while moving (B)
Background	Uniform (1)	Cluttered (2)

Table 2 - 4 videos in all. See video 32-35 for results.

Results:

A1) The simplest test is to rotate the object in place on a uniform background. The result is decent, but once again we see that is slides slightly of center. I believe that this is again due to the scaling error. As the globe rotates back right, only the points in the right side of the bounding box can be tracked (as the points in the left side is on the blank background). This leads to the earlier mentioned rightward drift. The scaling is perfect. The bounding box shrinks during the rotations and inflates to perfect size again.

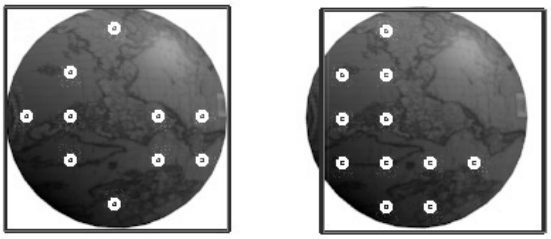


Figure 55 – A1: rotation on a uniform background. The resulting position is off center due to the unstable scaling.

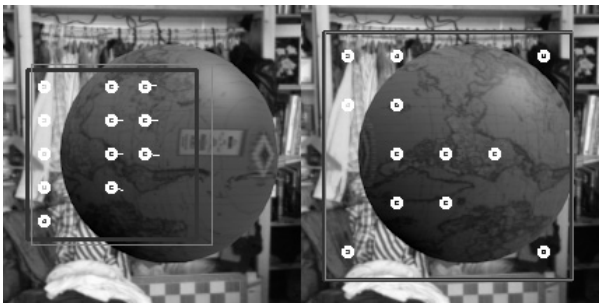


Figure 56 – A2: rotation on clutter. The points and the background are erroneously included in the calculated scale change leading to an overinflated bounding box.

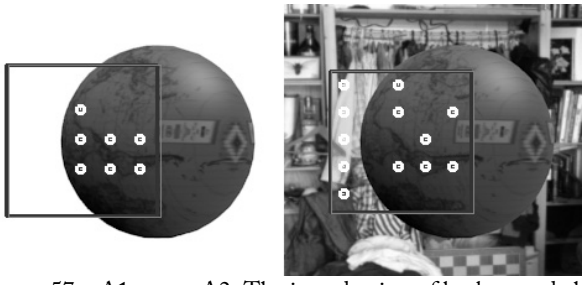


Figure 57 – A1 versus A2: The introduction of background clutter inhibits the tracker from moving freely as they are votes against further movement.

A2) If we introduce a background things get more complicated. The resulting bounding box ends up being too large and of center as can be seen on the right most picture below. The reason can be seen in the leftmost picture (figure 58): as the object rotates, the tracker includes background. The points on the background are included when calculating the scale change, even though they are not located on the object. Since these points are stable, but the globe rotates

right, the tracker thinks of the point distance as increasing and the object as enlarging, leading to an over inflated bounding box. The background further has the effect that it prevents the tracker from sliding too much off the object (which in some situations might be a nice quality, but could also be a problem) as the background functions as a wall. Once the tracking points are partly located on the stable background, they are a vote against further translocation. If we compare the two videos (A1 and A2) midway, we see that when no background is present the tracker is halfway off the object and there are no used points on the background. When introducing clutter, there is 5 points on the background and the tracker hasn't moved as much off object. With no background the tracker is freer to flow, as a background picture function as a movement-inhibiting wall.

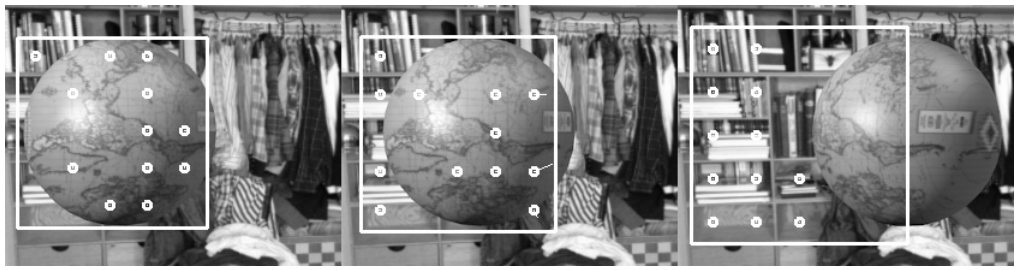


Figure 58 – The tracker is peeled of by an object that rotates and moves in opposite directions. See video 35.

B1-B2) Rotation and movement on a uniform background is tracked successfully with little uncertainty. However, the introduction of clutter completely alters the picture: The track runs cold after a few frames. It is the combination of movement, rotation and background that becomes too complex to handle. The tracker is peeled of as a sticker. If we track B2 backwards we get near perfect tracking. We can confirm this result by looking at actual recordings.

Rotation out of plane II

To test whether the problems seen in the synthetic experiments hold in real life, three videos are recorded.

Setup: A matryoshka doll is rotated 70 degrees left and then back while 1) standing still, 2) moving left or 3) moving right. Based on the synthetic sequences we can expect in place rotation to be fairly exact. Moving left to have a good position but too small and moving right to fail completely.

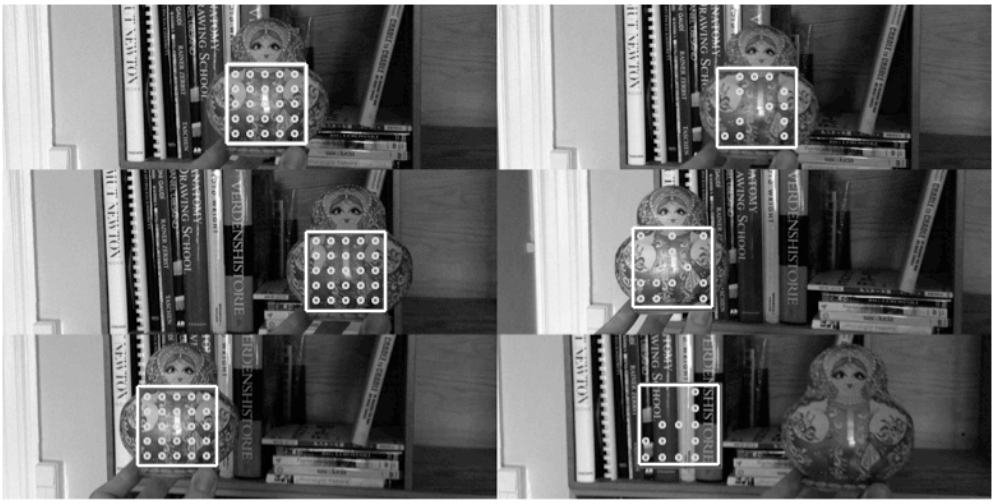


Figure 59 - Top: in place rotation. Middle: Rotating and moving left. Bottom: rotating and moving right. As in the synthetic experiment, the tracker cannot handle the object rotating left and moving right. See video 36-38.

Results: In place rotation performs very well and there is in this case no problem with scaling. Sequence two (moving left) proves to be very precise as well and the final position and scale are both good. Leftwards rotation while moving right fails exactly as we have seen in the synthetic experiments (results can be seen in video 36-38). The problem of rotation and movement is real in actual recordings as well. In all three recordings the tracker performs slightly better than in the synthetic sequences, but these were also designed specifically to break the tracker and as such it makes sense. The most distinctive error is preserved throughout. So to throw the tracker of I need to talk a step to my left while rotating my head to the right.

SUMMARY

The above tests, synthetic and actual, have demonstrated some characteristics of the median flow tracker. The proposed tracker works very well in many cases, but there is also cases where it has difficulties and fail.

The tracker works best with:

- Patterned objects. This is in contrast to objects that are primarily defined by color and/or form.
- Sub surfaces where little disturbance from surroundings is present. If the surroundings move in the same direction, it will disturb the tracking less.
- Rigid objects (though is handles deformations well as long as proportions are constant).

- Compact objects (no transparency and distributed mass). If the object is dispersed, the bounding box will include more background and this could create a problem for the tracker. Transparency leads to unstable intensities and hence tracking.
- Matte objects. Reflections can create large disturbances in surface illumination as we saw with the book.

It has difficulties with:

- Keeping position while scaling.
- Occlusion. More than 50% occlusion is likely to lead to tracking failure (unless it is field of view occlusion).
- Large of amounts of background included in the bounding box.
- Changing aspect ratio. This means that we cannot track oblong objects.
- Rotation and movement at the same time.
- Changes in exposure time and other sudden changes in illumination.

With respect to initially mentioned challenges we see that:

- Object change is handled well as long as they are incremental. This is the benefit of choosing an adaptive tracker.
- Change in angle (out of plane rotation) is more difficult, but can be handled under some circumstances (little background and/or no movement).
- Occlusion is difficult and tracking fails at around 50% occlusion. This is the disadvantage of being adaptive.
- Illumination changes are handled well as long as there is texture and the change is incremental. Exposure change, and other sudden brightness changes, leads to failure.
- Scale is handled decently if there is a uniform texture, but the tracker has a tendency to slide of target when scaling and rotating.
- Similarity is not a problem unless the object is close by. This could be fixed using FB tracking, but would result in doubling the tracking time (Matas & Vojir, 2011).

So far we have looked at the general challenges of tracking and classification. We have introduced the theory behind the TLD framework and its two components. And we have tested the median flow tracker. We now have an understanding of how well the tracker works. Next step is to understand the classifier.

CLASSIFIER EXPERIMENTS

In the following I will focus on what happens in the forest alone (not using any additional qualifiers), that is: how the forest, trees and features act. If the forest does not recognize a patch, it will never get to the last validation anyway. The full classifier will be tested on actual videos when comparing with the TLD framework later on.

FEATURE STATISTICS

Failing features

The features are chosen at random, but it would be interesting to know if some are more likely to cause classification error than others. As everything starts at feature level, I will look at feature stability.

Setup*:* Using 10 images, 10.000 sub patches are extracted. For each patch a random feature is generated, evaluated and compared to the same area evaluated with noise or increased brightness (a minor change that ideally is inconsequential). The features are simplified versions of the one described earlier: they only compare left and right side (not top and bottom) and thus only have two outputs. The intensity difference between left and right half is normalized with the features area size and saved as either an example of identical or different feature evaluation. In other words: what is the intensity difference for features that evaluate as the same versus those that do not. This is done to see if intensity difference is correlated with feature stability.

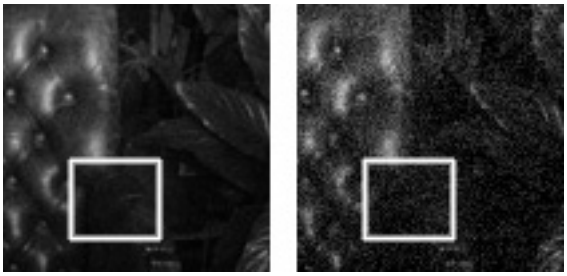


Figure 60 - Introducing a small amount of noise, the classifier fails as balance shifts in fragile areas. In this case the feature located on the armrest change value.

Result: The intensity difference between correctly identified patches are on average ≈ 12 (on a 0-255 scale). For failures it is 0.24 on noise and 3.18 on brightened images. This show us that there - with both kinds of images – is a large average intensity difference between cases where the feature is evaluated correctly and erroneously. When failures occur, the feature intensity difference is – with little variance – low. And successfully identified features have an average high intensity difference. But unfortunately there is a high variance, meaning that we do not have a failsafe way of recognizing the features that will

fail. All we know is that if the feature isn't recognized; it is very likely to have a low intensity difference. But having a low intensity difference is not the same as guaranteed failing. Low intensity difference is a necessary but not sufficient condition. Next experiment will investigate whether this has an influence when evaluating slightly translated patches.

Noise	Average	Standard deviation
tp	11.45	12.58
fn	0.24	0.24
% fn		2%
Brightness	Average	Standard deviation
tp	11.98	12.93
fn	3.18	2.38
% fn		11%

Table 3 – Table shows relation between true positives (tp), false negatives (fn) and intensity differences between feature halves. When features are evaluated differently it is correlated to a lack of intensity difference in the original image. When there is little difference, small disturbances might shift the balance, resulting in a different evaluation. But as the standard variation tells us, this is by no means a sure way of identifying the cases where errors occur. It seems to be a necessary condition, but not a sufficient. 11% of all features fail on brightened patches.

Setup and result: As before 10.000 features are chosen at random from 10 images. The feature output is compared with the output of the moved feature. The feature is moved up to 10% of its size. Again we see that robustness is correlated with intensity difference as false negatives drops when the threshold is increased.

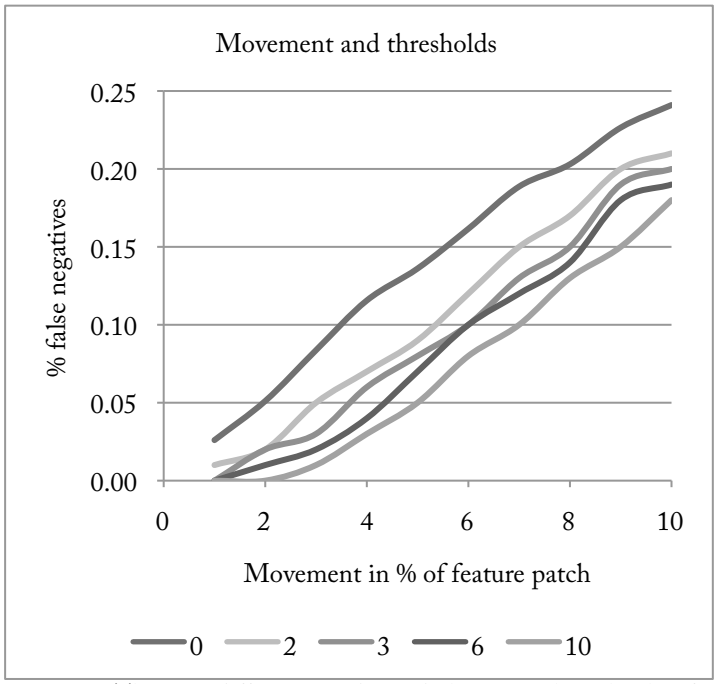


Figure 61 - The more difference we demand, the more robust the classifier get to movements.

Discussion: Small disturbances are introduced in all experiments, either explicitly or through rounding and loss of information. This means that the features, where the balance between dark and bright is subtle, are likely to change value. If the left side has a value of 1000 and the right a value of 1001, introducing small amounts of noise likely change the balance. And since many features are in near balance, exposure and other noise factors can have a large influence. If we filter out the features with initial intensity difference below 1, the failure rate falls from 2% to 0.01 % when analyzing pictures with noise. If we filter out features with a difference below 6 on pictures with increased brightness, failure rate drops from 11% to 2%. So thresholding the features obviously work well, but unfortunately this also means discarding many features. With a criterion of discarding features with an average pixel intensity difference below 6, 50% of all features are discarded – many of which did not contribute to the failure rate. And for this insight to be useful we need to handle the fact that contrast areas change depending on light and viewing angle.

Feature-Tree-Forest

What are the relation between feature, tree and forest? The forest – as we have seen – consists of trees, which in turn is a collection of features. A tree returns true when queried, if all features does so. A forest returns true if a majority of the trees does. So if we know the probability of a random feature not being a false negative we can give an estimate on the tree' and subsequently on the forest' answer not being a false negative. Assuming independency between features, a depth-10 tree with a feature error rate e is likely to have $1-(1-e)^{10}$ likelihood of false negatives. Turning to the forest (nine depth-10 trees) we can calculate the expected probability of the majority of n trees giving a correct answer by the binomial distribution. The binomial distribution is the probability of obtaining exactly k successes from n individual trials that each return either true or false (a so called Bernoulli trial). So given n trials – where each is an independent Bernoulli trial with probability p of success – the chance of exactly k successes is calculated using the probability mass function

$$f(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

is the binomial coefficient. Since we are interested in more than half being a success, we need to add the individual probabilities. Using 9 trees this equals the summing the probability of 5,6,7,8 or 9 trees giving a correct answer. The

underlying assumption is that the features are independent. This is the assumption that I will test in hope that it will expose parts of the relation between the three.

Setup: A small experiment is conducted to test relation between feature, tree and forest. 10.000 patches are chosen at random. For each patch a random feature, tree and forest is generated. The forest and tree is grown. The feature value is noted. The patch is the moved 1% and 5% and forest, tree and feature queried. If the response changes from the grown the answer is marked as a false negative.

Transformation	Feature	Tree (actual/predicted)	Forest (actual/predicted)
Moved 1%	0.04	0.33 / 0.34 fn	0.19 / 0.15 fn
Moved 5%	0.18	0.81 / 0.86 fn	0.91 / 0.996 fn

Table 4 – False negatives and the relation between feature, tree and forest. Our predictions based on feature performance are decent, but also tells us that the assumption of independent features is not perfect.

Results: The assumption seems to be acceptable within the tree and the relation is straightforward. A feature error of 4% false negatives ought to generate a tree with 34% false negatives, but in reality it is 33%. However, the correctness of the assumption is dependent on the transformation and amount. If we translate the image 5% instead of 1%, we get an individual average feature probability (of false negatives) of 0.18. In theory this will result in a depth-10 tree with 0.15 false negatives. But in reality the result is 0.19 false negatives. A feature with 4% false negatives should result in a forest with 15%, but actually have 19%. And a feature with 18% false negatives should create a forest with 99.6%, but in practice it is 91%. This indicates that the assumption no longer holds when looking at the whole forest (even when the translations are small). The features can no longer be considered as independent. Using 9 trees there seem to be covariance between some trees. This makes sense as the trees are likely to use features that are the same or close to the same, when many are generated. So when one succeeds it is likely that others will also succeed and vice versa. This means that we get less value per features, than if they been independent.

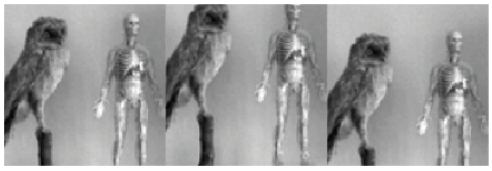
MOVEMENT, SCALING AND ROTATION

Robustness to position, scale and rotation is very important when scanning a frame. If the classifier is very sensitive to position, we have to query at all position, making it an expensive operation. Likewise it is expensive to scan at all scales and rotations. And finally, translocation, scale and rotation can be seen as simulations of minor object transformations (moving the jaw slightly, rotating or tilting a tidbit). Ideally a classifier can handle such transformations.

Setup: 3 pictures of interiors are used. In each picture 24 square sub patches are chosen. Each of the 72 patches is tested individually. The patch is grown and the classifier is then queried with slightly altered patches. The transformations are

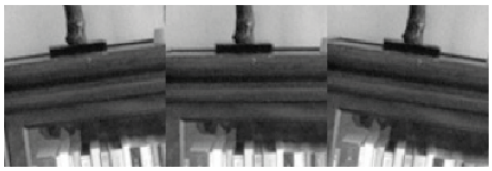
- Movement: queried region is moved up to 10% of patch width.
- Scale: queried region is scaled ±10% of patch width (at 90, 92, 94, 96, 98, 100, 102, 104, 106, 108 and 110 percent of grown patch).
- Rotation: queried regions are rotated up to 8 degrees.

In all cases the forest is composed of 8 depth-10 trees. The tests are run with eight different random forests. Movement and scaling is further tested with 10 degrees of blurring to see if this has influence.



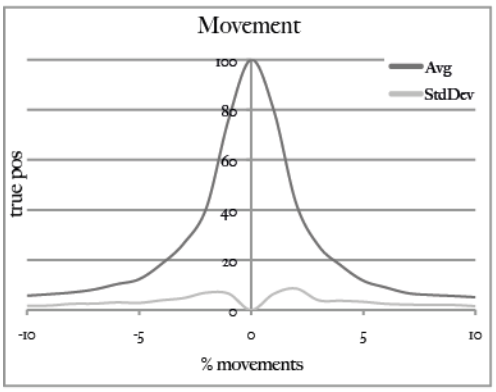
Movement

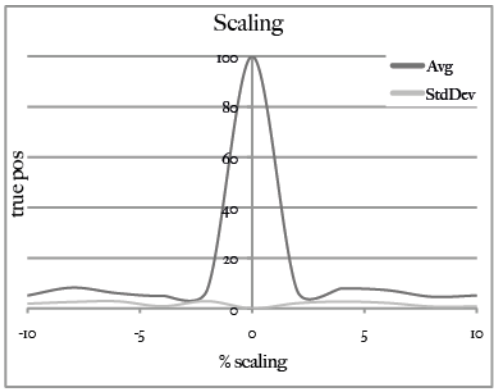
Scale

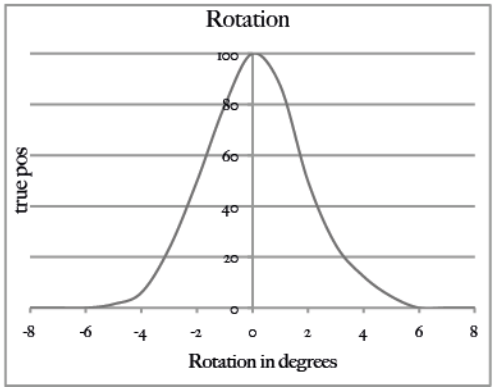


Rotation

Blur







Results: The classifier is fairly fragile to all three transformations. In case of movement, true positives drops to 50% with a translation of 2%. Scaling more than a few percent and there is practically no true positives. And if we rotate more than 5 degrees nothing is recognized. The difference between blurred patches and not is miniscule as we can see from the standard deviation. In case of movement, performance is slightly better when blurring, but nothing dramatic. And blurring further has the effect of increasing the risk of false positives.

Discussion: A way to solve this is to grow the initial region with affine transformations. This will create a more robust base, but subsequent additions to the model will suffer from this lack of robustness to small transformations when scanning for the object. Subsequently grown patches are, however, rarely grown isolated as they are chosen as a part of a tracked sequence. This could mean that the patches are naturally varied and transformed. And so this need for variation might be solved naturally by the growing strategy. That the classifier fails on these transformed patches is understandable given the features. The smallest features are 20% of image size. Moving the image 10% left thus means that these features will have changed completely: what before was in the left side of the feature is now on the right. So though it would be nice, it is only natural that the classifier cannot handle such large transformations.

CIRCUMSTANCES

Occlusion

The classifier is not robust to occlusion. This makes theoretical sense and can easily be demonstrated with a small experiment.

Setup: Using 240 images, each is grown and subsequently queried with varying degrees of occlusion. The occluding object takes up the entire height but varies in the horizontal direction. The maximum occlusion is 25%. The images are chosen at random from the Internet. In all cases the classifier is comprised of 8 trees with depth ten. The occluding image is the same in all cases. No smoothing is used.

Figure 62 - 0, 15 and 25 % synthetic occlusion.

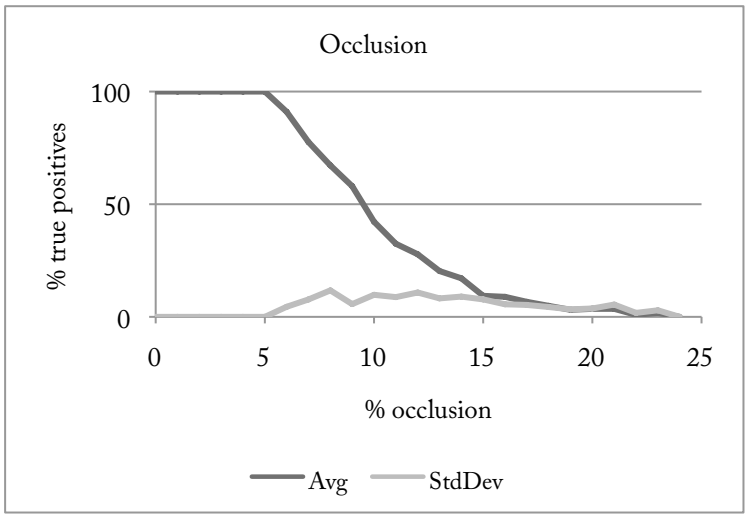


Figure 63 - true positives as function of occlusion. Up to 5% occlusion is handled well, after that performance drops sharply. And with 10% occlusion the object is only recognized half the time.

Results: The result shows the average performance of the patches. Though there is some variation between patches, the overall trend is clear: the classifier is not very robust to occlusion. More than a few percent and performance falls markedly. This makes sense given the design of the algorithm. More than half the trees must recognize the patch and all features in a tree are required to be confirmed before the patch is recognized. Put otherwise: if more than half the trees have a feature that involves the occluded area, there is a good chance the classifier will fail (given that the occluding object change the feature output). And since there are 10 randomly created feature patterns in each tree, the occluded area is likely included in the tree. The classifier is – in other words – sensitive to occlusion. It is possible that the classifier rejects the patch even though 93.75% of the features evaluate correct (5 features out of 80). This is because of the cascade structure. Only 50% of the features need to be correct, but they need to be grouped in trees.

Illumination direction

The features used by the classifier are in some sense robust to lighting changes and in another sense not at all. A small local area on the object is likely to retain its feature value across light changes if there is texture. The larger features, spanning over areas that are not in the same plane – such as from the left side of the nose to the right side – are likely to be influenced by changes in lighting conditions. We can see the value of adding texture with a simple experiment.

Setup: A synthetic object is created in a 3d-simulator. 6 different degrees of texture is used – starting at none. Two pictures with different light are taken of each. 1000 random feature areas is then selected and compared in the two images. The average number of stable features is registered. The goal is to test

whether surface patterns have any influence on robustness to changes in lighting direction.

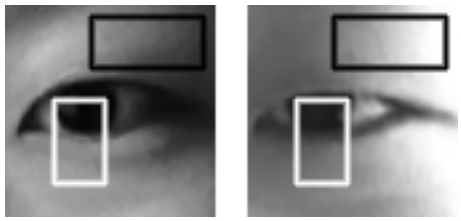


Figure 64 - Features placed on uniformly colored areas (black) are more likely to be influenced by illumination than those describing pattern change (white).

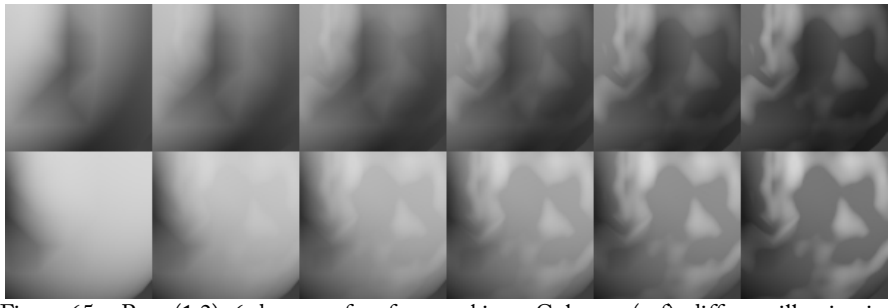


Figure 65 – Row (1,2): 6 degrees of surface markings. Columns (a-f): different illuminations of the same patch.

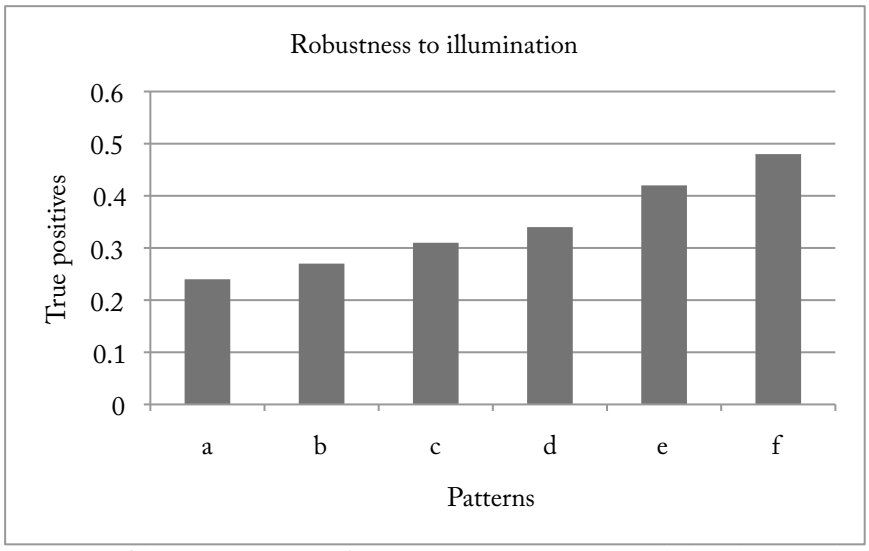


Figure 66 – Objects with more surface pattern variation are more likely to be correctly recognized across different lighting conditions. Smoothing and equalization makes little difference.

Results: The numbers clearly shows the positive effect of adding a surface pattern. Unfortunately we also see that even with a high degree of pattern, the average feature recognition rate is 0.48. This means that a depth-10 tree has 0.0005 chance of recognizing the highly patterned object. Running the actual test we get a better – but still useless – result of 0.001 correct answers. Growing the tree using affine transformations (scale and position) does not help to

improve this result. The classifier is – in its current form – sensitive to changes in illumination direction. It helps if the object has a rich pattern, as this property is light independent, but if the light changes markedly the classifier fails utterly.

Cropping and background

When analyzing the tracker we saw that, though it preferred no background interference, it were fairly stable even when background were included. How well does the classifier handle background interference?

Setup: The exact same synthetic object is placed on two different backgrounds. The object along with a part of the background are grown and subsequently queried on the other image using the forest (9 depth-10 trees). The test is run with 10.000 different forests to ensure that this is not a question of unfavorable randomization.

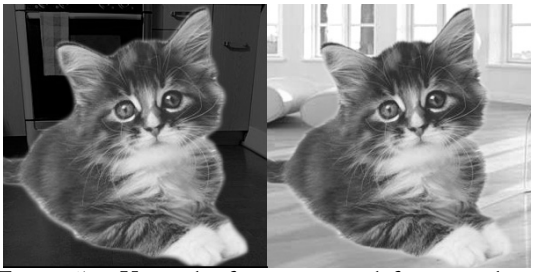


Figure 67 - Using the forest to grow left image, the right cannot be recognized due to the background clutter even though the cats are identical.

Results: The patch cannot be recognized a single time. The included background creates too much noise. If we draw the recognized features we get an idea of what happens and which areas are recognized. Mainly features located on the actual object are recognized, just as hypothesized. But as the forest consists of features distributed across the entire patch, the forest is highly likely to fail when a changing background is included in the model.

If background is included when growing, this background is also expected when querying. The classifier makes no assumption about the objects layout and thus treats all pixels within the patch equally. So a feature might be located on the background entirely and will thus be a way of recognizing the background. Or it can be located partly on the background and partly on the object. In this case it is the intensity relation between the two that are registered, so if the object is brighter than the background, this relation is a part of the model. But this relation might change if the background changes. Basically background changes can be viewed as occlusion. Some part of the grown model is not present, but replaced by something before unseen. This leads to recognition failure. Adding the same object multiple times with changing backgrounds will not change this.

It will simply lead to multiple branches, but every time a new background is introduced, the features will change and the classifier fails.

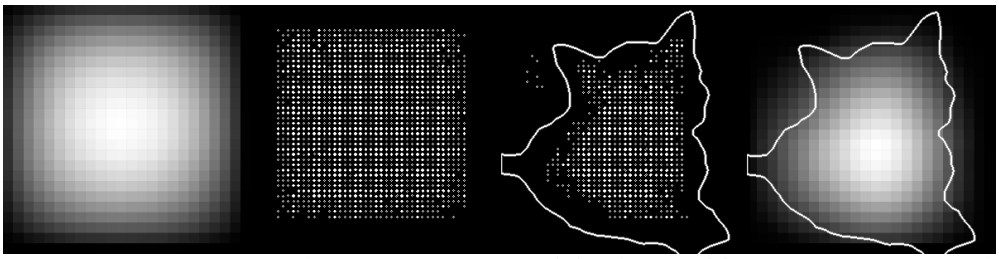


Figure 68 – First two images show average coverage and distribution of feature centers. Next two show those features that are recognized. The white lines are added manually to show the contour of the kitten.

Categories versus particulars

The system is geared toward learning particulars and not categories. The classifier might just as well learn a category as a particular, but the interaction between classifier and tracker is designed for particulars. This is so because of the pruning process. We prune patches from the model whenever we believe that we confidently have tracked the object. This means that if there are two instances of the same category present in the image, we cannot learn this category. Instead the classifier will learn what makes the tracked instance different from the remaining image, including the other instances of the category. This could be solved by not pruning, but as occasional errors will occur during growing, this will lead to a degenerating model. This means that pruning is essential for performance (Kalal, Matas, & Mikolajczyk, Online learning of robust object detectors during unstable tracking, 2009). And the object might – at some views – be indistinguishable from clutter and thus be pruned from the model as being too uncertain.

SUMMARY

We now have a better understanding of both classifier and tracker. Both prefer compact objects with are rich on surface markings. And both prefer sub patches to whole objects since the background interferes. The classifier is sensitive to small transformations such as movement, scale and rotation. Affine growth or more elaborate scanning can solve this problem. Neither handles occlusion well, which could be a problem in some situations (tracking a moving and partly occluded object). Both tracker and classifier are also sensitive to illumination. The tracker can handle this well if the changes are small, but a large change will lead to tracking failure and the classifier will not be able to initialize anew. And finally do both prefer rigid objects. Neither can handle a heavily deformable object as a scarf or even a scissor; the tracker will fail on bounding box and background clutter. The classifier on background and the ever changing object configuration.

So ideally we track and classify something rigid with a pattern and not something defined primarily by form and color (e.g. spoon or orange) or relation of movable parts (e.g. scissor). Further both work best under stable lighting and with no occlusion. The TLD framework should be able to handle some degree of changes in light and transformations as long as they can be tracked, since they are then added to the model if all goes well.

We now have an idea of how well different challenges are handled:

- Object change is not well handled. Even small changes in the patch leads to classification failure. We have seen this both, when slightly moving the patch and when experimenting with background inclusion. This means that small changes such as object transformations leads to recognition failure. This is also a problem for out-of-plane rotations. To handle this we need to add the new views of the object to the forest.
- Illumination changes are handled well by small features, but the classifier as a whole fails when the light changes. Having a rich texture helps, but if the change is dramatic it will not be enough. This means that the patch needs to be grown under different illuminations.
- Position and scale is handled by growing transformed versions of the patch. Without it, even small transformations lead to classification failure. Position and scale is otherwise handled by the scanning procedure.
- Noise can have a great influence on the delicately balanced features, and thus on the whole classifier. This can somewhat be accommodated by affine growth and thresholds.
- Very similar patches will with the current framework lead to growing and pruning of the same branch repeatedly.

TLD EXPERIMENTS

We now have an understanding of what our tracker and classifier can and cannot. Both seem to prefer patches that are a part of a larger object and that have pattern. Neither likes occlusion and exposure- and light changes on uniformly colored objects. However, combining the two we should be able to recuperate from occlusion. And if the object have pattern, we should be able to track through many lighting changes and possibly add these patches to the model for future recognition. Exposure is a major problem for the tracker, but the classifier might be able to handle this. The problem of minor transformations can be handled by tracking, as the classifier will only have to recognize patches when the tracker fails. It might in some cases take a little longer, but when it happens, the tracker will handle the minor transformations.

Major transformations will ideally be added to the model if the object is tracked successfully throughout and returns to a familiar appearance.

FOCUS AREA ON EYES

For the testing of the interaction I will choose a specific application where I believe the framework might work as well as be challenged: eye tracking. The eyes and the areas around have plenty of intensity variation. They are surrounded be matter that moves correspondingly (the remainder of the head). And at the same time they undergo large appearance transformations and have cavities that mean that they change appearance with lighting. Further hands, turning of the head etc, regularly occlude them. An example of where this could be useful is as preprocessing to gaze estimation. Further they are a good representative of the general challenges involved in tracking, learning and classification.

TEST MATERIAL

Two types of test material are used: videos of eyes and videos of objects. The main interest is the eyes, but having other and more diverse objects as a backdrop can informs us whether our parameter changes only have value to a specific kind of challenge, namely eyes. 5 subjects are given a list of actions to perform while holding a recording mobile phone. These actions are not to ensure that the subjects do the exact same thing in the exact same circumstances, but merely to ensure that all the challenges involved is represented. Each video is roughly 2 minutes long and the instructions as follows:

- Look at your self at medium distance.
- Move you head slightly within the frame.
- Stretch your arm to move the camera farther away.
- Move to close-up.
- Medium distance and large movements within frame.
- Fast sudden movements.
- Tilt head to side.
- Look around with eyes only.
- Close eyes.
- Move phone to stomach region.
- Move camera to the side.
- Look at phone while it is over head.
- Turn 180 degrees (to change light).
- Touch eye with finger.

- Occlude eye with hand.
- Occlude and move to new position in room.
- Move around at will.

The subjects are of different race and gender and recorded at different place and hours.

1) Male Caucasian with glasses at daylight.
2) Male Caucasian with glasses at night.
3) Female Asian at daylight.
4) Daylight recording of female Caucasian with hair below eyes.
5) Male Caucasian at daylight.

All videos are recorded in 640 x 480 pixels with a frame rate of 12 frames per second. Ground truth has been manually denoted in all frames.

Figure 69 – First frame of the 5 baseline videos and at bottom right the subject occasionally used for parameter estimation. See 'TLD test'-folder for videos.

Figure 70 - Frames taken from one of the baseline videos. Angle, field of view occlusion, changing light, distance, motion blur, shadows, occlusion, transformations, reflections, other subjects and large amount of noise due to low lighting is all represented in this video. See 'TLD test'-folder for video.

The object videos are: two books, a charger, doll, beer, two toys and a vase. They are all moved in the same context and undergo scale, movement, occlusion, light change and disappearance. Finally a sixth subject's eye is recorded. This video is occasionally used to experiment with parameters, the result of which will be tested on the above videos. This video is divided into segments for easy analysis of which transformations are difficult. Further a video of pure clutter will at times be used to test for false positives.

Figure 71 - 8 objects used for baseline comparison to test whether the framework can handle other types of objects than eyes. See 'TLD test'-folder for videos.


Figure 72 - Illustration of some of the transformations seen in the parameter test video: occlusion, light, angle, motion blur, rotation and internal object transformations. See 'TLD test'-folder for video.

Measurements: when testing several values can be of interest. Precision and recall is often used and so will I. Precision *P* is in this context defined as the number of true positives divided by the number of all detections made. Recall *R* as the number of true positives divided by the number of object occurrences that ideally should have been detected. Precision is in other words how relevant the

detections are while recall is how large a fraction was detected. A detection is considered a true positive if the area of the union with ground truth is more than half of the twos separate areas. The two results can be summarized using the harmonic mean (F): 2*P*R / (P+R).

COMPARING TRACKERS AND A BASELINE

We have so far looked at the median flow tracker and the forest classifier. The latter can also be used for tracking by detection – choosing the most likely (the one with highest template matching score) patch if multiple patches is detected. A reasonable first question is whether the combination proposed in the framework performs better than the individual parts. This result will also be used as a baseline for future comparisons.

Setup: we can test this be analyzing the videos using the tracker, classifier and framework. The median flow tracker is reinitialized after each part. Classifier and TLD analyze the video as a whole based on the first frame. The forest threshold used in tracking by detection is 0.5 when matching with the model. The threshold for growing is 0.7 and initialization is 0.8. The initial patch is grown with 500 simple transformations (position and scale only as this can be done very fast by simply adjusting the region of interest). And the tracker can be restarted if a detection is made with confidence above 0.8. Separate results are given for eyes, objects and parameter videos.

Baseline		P	R	F	First failure
Eyes	Tracker	0.31	0.12	0.16	120
	Classifier	0.90	0.26	0.36	6
	TLD	0.63	0.68	0.65	147
Objects	Tracker	0.81	0.28	0.37	124
	Classifier	0.96	0.57	0.69	43
	TLD	0.88	0.79	0.83	125

Table 5 - Average precision, recall, harmonic mean and frame of first failure for the 5 subjects and the 8 objects.

	Tracker	Classifier	TLD
Move (S)	1.00	0.91	**1.00**
Move (L)	0.96	0.91	**1.00**
Closed	1.00	0.94	**1.00**
Gaze	1.00	0.63	**1.00**
Blur	0.07	0.60	**0.74**
Rotation	**1.00**	0.29	0.97
Angle	0.31	0.42	**0.95**
Light	**1.00**	0.81	0.99
Occlusion	0.68	0.73	**0.81**

Disappearance	0.05	**0.74**	0.62
Hide - angle	0.07	**0.48**	0.31
Hide - light	0.06	**0.64**	0.44
Clutter	0.54	0.06	**0.68**
Average	0.60	0.63	**0.81**

Table 6 - Harmonic means for tracker, classifier and framework on the parameter eye video. Neither the tracker nor the classifier alone can compete with the combination of the two.

Results: The baseline videos shows us the overall benefit of the framework while the parameter video that is divided into transformation isolated segments give us an understanding of where the trackers have their strengths and weaknesses.

- The median tracker alone works well on some sequences, but fails on fast movements, out of plane rotation and occlusions. It handles light changes surprisingly well given what we earlier have learnt about the tracker. The explanation is likely the speed and pattern involved. The changes are continuous and slow (around 20 frames) and the tracked area has a fairly distinct pattern. Though the average performance of the tracker alone is 0.6, this is only because the tracker is restarted between parts. Since the tracker has no way of getting back on track any true positive after the first failure is pure luck.
- Tracking by classification has a better performance. Especially movements and occlusion is handled well. This makes sense as the object looks roughly as in the model. More suppressing is it that closed eyes and light changes is handled so well. The eye is closed in 31/91 frames. This of course is great in the current situation, but it could well mean that it would misclassify in another situation.
- Best of all does the combination work. On most parts it is at least as good and often better. An exception is the videos with total occlusion. The trackers inability to detect own failures can explain this. So when the object is suddenly occluded, the tracker will continue to track the occluded – exactly as we saw in the synthetic tests. The pure classifier does not have this problem. Further we see that the classifier alone has a higher precision, but lower recall, meaning that few patches are recognized, but they are done so fairly accurately.

FRAMEWORK VARIABLES

The framework has several parameters, which can be tuned for the specific purpose. One is how similar the two patches have to be before they are considered identical. In previous experiment we operated with a model threshold of 0.7. So any patch that were evaluated positively by the tree, but did

not have a NCC match in the model were discarded. What is the optimal value of this threshold θ? This threshold defines when we grow.

Setup: Precision and recall is recorded during testing with different values of θ: 0, 0.25, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9 (where zero meaning that everything is accepted and 1.0 that the patch has to be identical to a patch in the model. The threshold for initialization is still 0.8.

Results: surprisingly there is practically no difference whether we use template matching or not for growing. Average precision is in all cases 0.78-0.80 and recall 0.81-0.83. The only scenario in which it makes a difference seems to be when analyzing clutter where precision varies between 0.67 and 0.9. Recall remains the same. As this result is somewhat puzzling, it seems reasonable to compare the baseline with a modified version with a threshold of zero. Here too we see that there is practically no difference (less than 0.01 on the average harmonic mean). This leads me to conclude that the forest itself is such an effective filter, that adding template matching does little difference when the tracker has already confirmed the patch.

Detection

We have now looked at tracking, as this is where it all starts and how the classifier is constructed. The interaction with the tracker has been greatly improved by adding failure detection and we can now trust our tracker to a large degree, which means that the tracker rarely adds false patches to the model and thereby dilute its quality. Next step is to ensure that the classifier initialize tracking in a decent way. I will therefore look at the threshold for initialization.

Setup: Thresholds at 0.7, 0.8, 0.9, and 0.95 are tried. The videos used are two eye-videos and three object-videos. 228 frames of pure clutter are added at the end. This is to test how unseen clutter is handled, i.e. does it create false positives.

Threshold	P	R	F	False positives on clutter
0.5	0.64	0.71	0.67	0.30
0.7	0.70	0.74	0.71	0.19
0.8	0.70	0.72	0.70	0.10
0.9	0.65	0.65	0.65	0.01
0.95	0.50	0.45	0.47	0.00

Table 7 - The initialization threshold has a great influence on the number of false positives. But also the recall and precision rate change. Around 0.8 or 0.9 seems to be the optimal compromise.

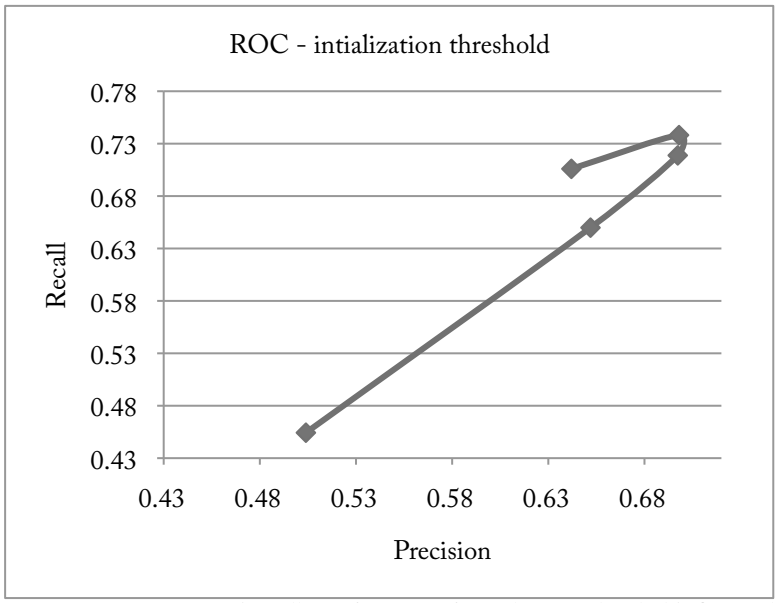


Figure 73 - Precision and recall as a function of initialization threshold. Optimal value is around 0.85.

Results: When to initialize tracking? So far we have worked with an arbitrary threshold of best above 0.8 using NCC. Testing this threshold there is no good answer. Recall and false positives both drops as the threshold is increased. An acceptable compromise might be a threshold 0f 0.8-0.9, but the problem seem to be, that the classifier is not good enough. We still have false positives, but if we raise the threshold further the overall recall drops.

MODIFICATIONS

One could imagine several ways of modifying the system. The stronger the tracker and classifier is, the stronger the framework. False and noisy patches can be added to the model in two ways. If the tracker tracks the wrong object and accidentally gets back on track, we risk growing this sequence. Occlusion can also be an issue here. If the object is partly occluded somewhere along the trajectory, there is a good chance we add partly occluded patches to the model. This might or might not be a problem. Another way dilution of the model might happen is if the classifier reinitializes at or identifies a wrong patch.

Further there is the question of growth. As it is now, we only grow if there is a loop on the trajectory, meaning that the classifier recognizes a patch on the trajectory. The intermediate tracked patches are then added to the model. A very careful approach would be to only add the trajectory when a patch matched our initial patch, which is the only one we know for sure, is true. If we knew for sure when to trust our tracker, we could grow more carefree (assuming that we

could trust the initialization be the classifier) most of the time and only require validation if the trajectory was uncertain.

As discussed when looking at the initial results, occlusion seems to dilute precision due to continuous tracking of the lost or occluded patch. I will therefore look at two possible tracker extensions: the first in hope that it will create more stable tracking, the second to detect failures.

POINT SELECTION

Matas suggests a different point selection strategy to supplement the median tracker (Matas & Vojir, 2011). Instead of initializing all points anew in each frame and doing so in a grid, they suggest allowing the points to wander from frame to frame. The idea is that the points will drift to areas that are easier to track and thus increase the quality of the tracked points. At such it is a new point selection strategy on level with Harris corner detection and other. Good points to track are points that trackers drift to. To ensure that the points don't drift away from the object or all end up on the same spot, the points are only allowed to drift within a restricted area. They call this approach "Cell flock of trackers", where the previous used method is a "Grid flock of trackers". Matas et al. experimentally shows that their approach is superior to the grid approach.

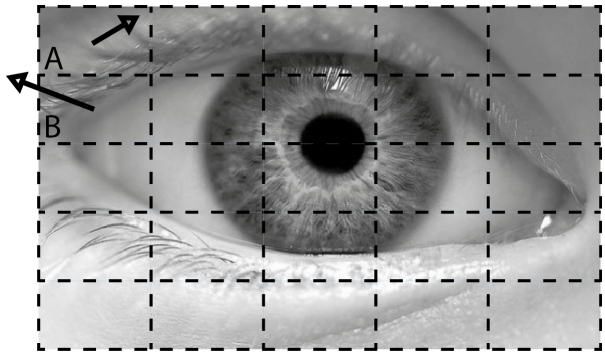


Figure 74 - Point selection. Points are allowed to drift during tracking, but if outside initial grid cell, it is reinitialized. In above A' new position is used for tracking in next image, while B is reinitialized to center of cell.

Setup: The divided sequence as well as the base sequences is tested with both trackers. The divided sequences are tested with 5 different forest initializations (the same in the two settings) and the baseline videos with a single forest configuration.

	Grid FoT	Cell FoT
Divided	0.81	0.80
Eyes	0.65	0.65
Objects	0.84	0.83

Table 8 - Whether we select points in a grid or allow them to float between frames makes little difference to the overall result.

	Grid FoT	Cell FoT
Eye - Kenny	266	**363**
Eye - Malthe	4	4
Eye - Sara	**19**	11
Eye - Sille	**95**	52
Eye - Thomas	72	**137**
Beer	**184**	118
Charger	37	37
Book1	76	76
Book2	80	80
Horse	317	317
Lego	104	104
Doll	123	123
Vase	54	54
Average	110	114

Table 9 - Comparing the two point selection strategies on first frame of failure. Bold numbers indicates best performance.

Results: The differences are very small. If we look at the individual parts, we see some variance. But the average performance is very similar. Concluding that on these sequences, the floating points selection adds nothing to the quality of the tracker, neither when used in combination with the TLD framework nor as stand alone tracker. This is in contrast with the results of Matas et al. that showed an advantage of their approach over the simpler grid selection. On explanation could be that they used NCC as an error estimate where as I have used SSD (according to Trucco (1998) the latter are more stable).

FAILURE DETECTION

Detecting tracking failures or giving an estimate on the quality of the tracker would me useful. Not having a perfect tracker is less a problem if you at least know when to trust it. Having such a tracker might remedy the false positives we see during occlusion and it might allow the TLD framework to loosen the requirements for when to grow. Further we might initiate another tracking approach when failure occurred, or verifying when insecure. I therefore believe that a tracker that – unlike the original median flow tracker – can estimate its own quality could possibly be very valuable.

Before calculating the median flow the 50% most unconfidently tracked points are rejected. These points are rejected based on the point flow tracking error. But there is no guarantee that these points actually agree on a transformation as the three parameters – horizontal movement, vertical movement and scale – is calculated separately and there might be outliers included in the used points.

And even with perfect point tracking will the tracker make an erroneous update at times due to the unstable handling of scaling. The tracker points may, in other words, result in a transformation, which none of them fit with.

One way to detect tracking failure is to match the successfully tracked points with their supposed transformation. To measure this distance the grid points on the new transformation is compared to the tracked points. The distance from each tracked point to its correspondent ideal points is measured. These strategies introduce two parameters: the maximum distance between point pairs and the percent of inliers α required. If more than α of the tracked and used points are within distance δ of the transformation, the tracking is deemed a success.

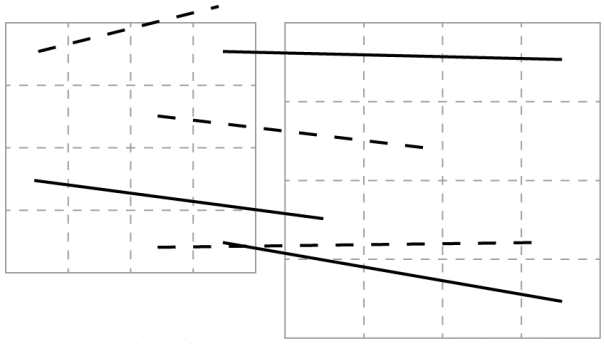


Figure 75 - If too few points is consistent with the resulting transformation, the tracking is considered a failure. Dotted lines denote tracked points that are inconsistent with the transformation. In this example only 3/6 of the tracked points is consistent with the transformation and the tracker has failed.

Setup: To test the quality of the new verifier, adjacent frames in the baseline videos are tested individually. The median flow tracker is initialized to the ground truth position between frames. The verifier it tested by noting whether it correctly verifies the proposal by the tracker or not.

- True positives: correctly tracked and verified.
- False positive: wrongly tracked and verified.
- True negative: wrongly tracked and rejected.
- False negative: correctly tracked and rejected.

The verifier it tested with thresholds 0.0 – 1.0 with 0.1 intervals. 0.0 meaning that no points need to fit the transformation and 1.0 that all tracked points need to match. The initial threshold distance δ is set to 1/5 of the diagonal (based on the usage of a 5 times 5 grid) during this experiment. To ensure that we do not simply optimize for these videos, parameter estimation is only performed on 2 eye videos and 3 object videos. The remaining 3 eye videos and 5 object videos are used for verification of the results.

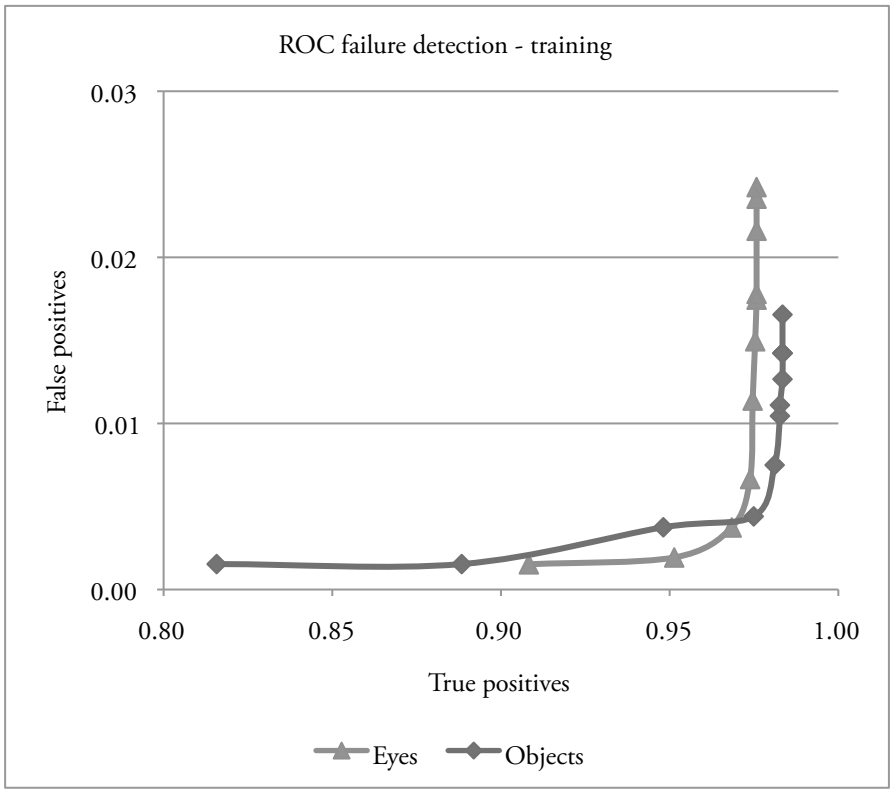


Figure 76 - ROC curves depicting the relation between false- and true positives as a function of inlier threshold. From this samples, the optimal threshold is around 0.7.

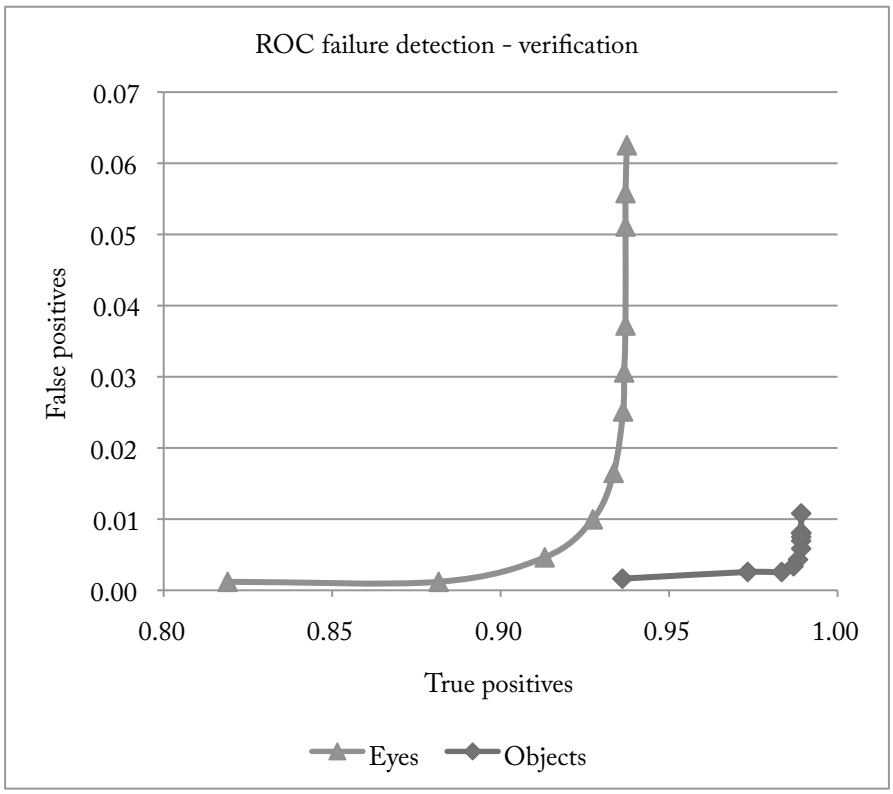


Figure 77 - Verification data confirms the parameter estimation. Here to is the optimal parameter in both cases around 0.7.

Results: The ROC curves show the relationship between true and false positives as a function of the parameter α. When demanding that all points are inliers, the number of true positives drops to below 90%. This also means that false positives are as low as 0.1%. Increasing the tolerance to outliers, number of true positives rises, but on the cost of false positives. For both objects and eyes, the optimal threshold is about 0.7, which corresponds to the points closest to lower right corner. Doing so decreases the number of false positives around 4.5 times while retaining the true positive rate. The next step is to adjust the parameter δ.

Setup: Setting the inlier threshold α to 0.7, I will adjust the maximum distance parameter δ. Starting with δ=1/3 of the diagonal and down to 1/10. Grid density is still 5 times 5 and the test material as above.

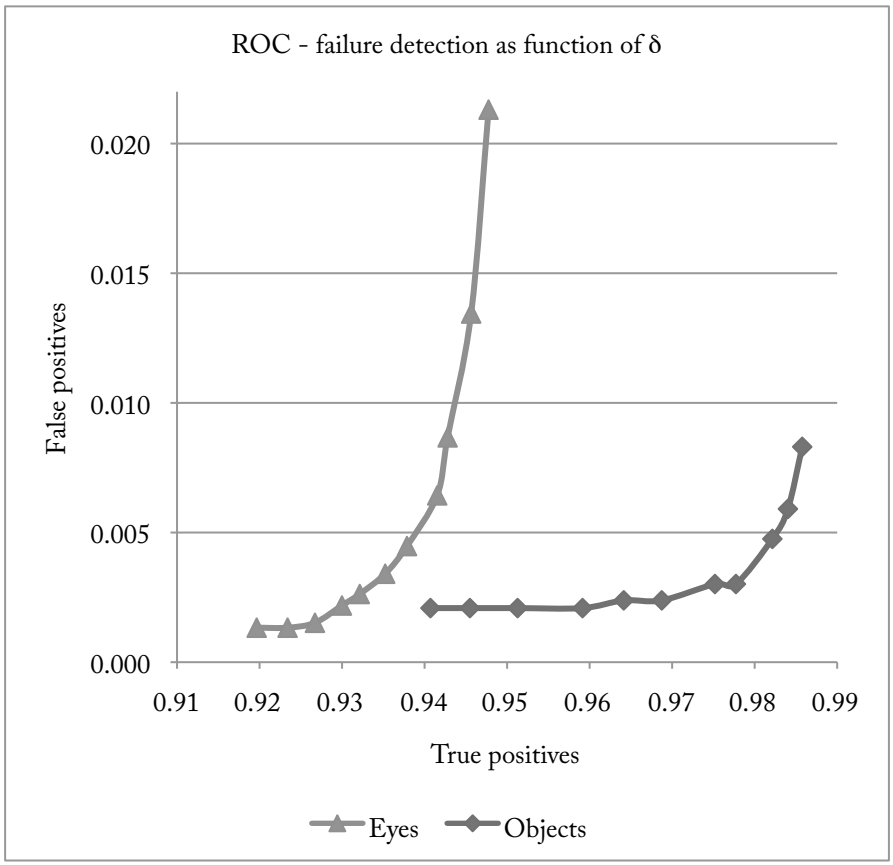


Figure 78 - ROC curve as function of distance threshold δ. The breaking point is around 20% of object diagonal.

Results: As before the ROC curve shows true- and false positives as a function of the sought for parameter. The curves are not as easily readable as before, but the optimal balance for δ seems to about 20% of the object diagonal. Incidentally this is also the value used in the previous preliminary experiment.

Based on these experiments, I will set the threshold α to 0.7 and distance δ to 1/5 of the diagonal. But before continuing with these values we investigate

whether there is any gain in using this more sensitive tracker in combination with the TLD framework.

Setup: The baseline videos are tracked using both TLD and the tracker in itself. We have earlier seen where the tracker fails using no failure detection. Ideally our new tracker will not fail before that and hopefully it will detect the tracking failure in this and other cases, leading to an increase in precision when testing with the TLD framework. Only one random forest configuration is used.

Sequences	Eyes	Objects
P (TLD)	0.89 / 0.63	0.98 / 0.89
R (TLD)	0.69 / 0.68	0.79 / 0.80
F (TLD)	0.77 / 0.65	0.86 / 0.84
First failure (tracker)	120 / 120	122 / 124

Table 10 - With-failure-detection / no-failure-detection. Adding failure detection have practically no influence on number of correctly tracked frames, but results in a markedly increase in precision.

Results: using the proposed failure detection rarely leads to discarding correctly tracked patches as can be seen from the almost identical number of tracked frames and the recall rate. The precision is greatly improved leading to way fewer false positives. Looking at the eye sequences alone, we see that precision rises from 0.63 to 0.89. And looking at the object videos we get almost perfect precision while retaining the recall rate. This tells us that our failure detection is useful. It might be a bit too harsh at times, since the recall rate and frame of first failure, drops slightly, but overall it works well. The failure detection can be expected to have problems with large degrees of rotation and perspectival transformation. This is because our bounding box is only allowed to scale and move with the current strategy.

Figure 79 - One place where the failure detection is too harsh and discards a correctly tracked patch is between these two frames, where the tracker correctly tracks the occluded beer label. One way to avoid this might be to lower the amount of points required for the transformation to be considered consistent. In this case only 8/12 matched and the result was therefore discarded.

Summary: introducing failure detection greatly improves the performance of the tracker and the TLD framework. It is also worth noticing that even with failure detection errors occur without being detected. One example of this is in above figure of the beer label. In this case the tracker succeeds in tracking the label, but fail to verify. In the context of the TLD framework, this might actually be an advantage as continuing to track might add an occluded sample to the model.

The parameter estimation would benefit from more thorough testing. Though the initial values were selected with consideration, it seems suspicious that the threshold δ is optimal at 20% since this was also the value used when testing for α. It could well be that the thresholds δ and α were interconnected and finding α based on another δ would have resulted in different thresholds. Further different rejection strategies could be used based on more information than just inliers and number of tracked points with an error estimate below the median. Rating tracking confidence could be useful for several things. We might for example initiate an additional tracking procedure in an attempt to correct the mistake. Or we might use this newly found confidence to create a different growing strategy. Or we might simply use at a way to reject uncertain patches as in above experiment and thereby increase precision and decrease risk of adding false patches to the model. In remaining experiments this new tracker will be used.

FEATURE RESOLUTION

Kalal et al. suggests NCC template matching with a 15 x 15 template and a 24 x 24 resolution on our features. If we look at the bear label and its corresponding view in the model, we get an idea of why the classifier fails. What makes the beer label special is simply not captured in such a low resolution.

Figure 80 - From left to right: The original. How the forest sees it. How the model sees it. This resolution is fine for some objects, but for the beer label, which ultimately is recognized by its text, it does not suffice. Of the 5 objects tested during 'detection' it had by far the most false positives.

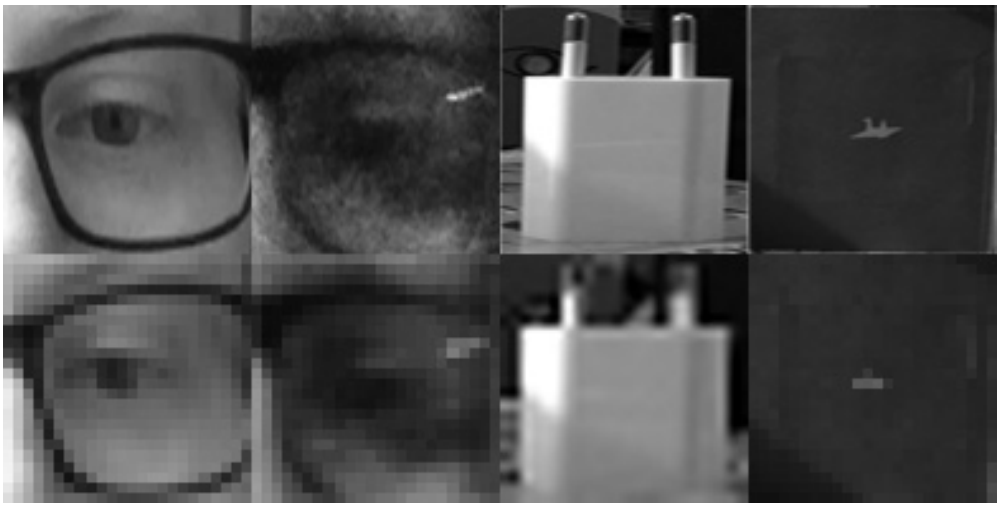


Figure 81 - The 4 other test objects and how the forest views them. Though there is still lost much information, the overall understanding of the object is better conveyed in this low-resolution view.

One way to handle this is to increase the resolution of the forest and the model. Sadly, increasing the model resolution to e.g. 50 x 50, will also make model matching much more expensive. Increasing resolution of our features is free of charge.

Setup and results: Using the resulting parameter setup from previous experiment a new test is performed on the divided eye sequence and two full sequences (one of an eye and one of a beer label). Three constellations are attempted:

1) Baseline (0.8 threshold for initialization, 0.7 tracking confidence threshold, 0.7 forest acceptance threshold and 15x15 template matching and 24x24 feature resolution).
2) As 1, but with a 100 x 100 feature resolution.
3) As 2, but with 50 x 50 NCC template matching.

Challenge	Normal	High resolution feature	High resolution feature and template
Move (S)	1.00	1.00	1.00
Move (L)	1.00	0.99	1.00
Closed	0.99	0.98	0.97
Gaze	1.00	1.00	1.00
Blur	0.97	0.97	0.97
Rotation	0.98	0.96	0.94
Angle	0.92	0.93	0.95
Light	0.98	0.99	0.99
Occlusion	0.89	0.89	0.89

Disappearance	0.92	0.93	0.92
Hide - angle	0.64	0.65	0.68
Hide - light	0.75	0.75	0.79
Clutter	0.66	0.67	0.69
Average	0.90	0.90	0.90

Table 11 - Changing resolution makes little difference when it comes to tracking in these sequences.

Eye/Beer	F24	F100	F100 + T50
P	0.87 / 0.81	**0.91 / 0.83**	0.90 / 0.81
R	0.85 / 0.94	**0.86 / 0.97**	0.86 / 0.97
F	0.86 / 0.87	**0.88 / 0.89**	0.88 / 0.88

Table 12 - Selecting two random videos we see a slight improvement when using a finer feature selection resolution. Using a larger template for NCC makes little difference.

The differences are small, but there is a small improvement when using a larger template resolution. Following I will use the new feature resolution of 100x100 as this seems to slightly improve performance and has no influence on speed. Increasing the template size has little influence on the quality and is slower. I will not use this in preceding experiments.

FEATURE SELECTION

So far we have chosen the features at random. But as we saw when during testing of the classifier, features that had little contrast where more prone to failure than those placed on contrast rich areas. Analyzing the initial image for these regions and using features located in these areas might improve performance. But as we operate with a flexible model that can accommodate multiple views these regions might change during tracking. But with the current framework where features are generated once, we cannot take future views into account.

Setup and result: A simple way to select features is to generate twice the amount needed and use the 50 percent that has the highest contrast on the initial image. Doing so on the 13 videos we see an increase in performance. Most are roughly the same, but there is a large performance increase in case of the charger. This object has absolutely no texture, making it ill suited for our classifier as well as tracker. But seemingly choosing contrast areas for features helps a lot. This makes sense since there are so few features that are interesting. It is really only along the edges that there is information unlike the other objects where we can be more careless with our choice of features.

Contrast selection				
Video	P	R	F	False positives on clutter
Eye Kenny	0.90	0.78	0.83	0.03
Eye Malthe	0.91	0.77	0.83	0.14
Eye Sara	0.85	0.78	0.81	0.05
Eye Sille	0.88	0.65	0.74	0.05
Eye Thomas	0.87	0.76	0.81	0.00
Object beer	0.95	0.99	0.96	0.13
Object charger	0.93	0.74	0.82	0.03
Object flying	0.93	0.60	0.73	0.00
Object hacker	0.92	0.92	0.92	0.01
Object horse	0.98	0.75	0.85	0.00
Object Lego	0.84	0.66	0.74	0.00
Object troll	0.75	0.92	0.83	0.46
Object vase	0.89	0.89	0.89	0.03
Average	0.89	0.79	0.83	0.07

Random				
Video	P	R	F	False positives on clutter
Eye Kenny	0.84	0.70	0.76	0.00
Eye Malthe	0.90	0.67	0.76	0.00
Eye Sara	0.84	0.79	0.81	0.13
Eye Sille	0.88	0.59	0.71	0.00
Eye Thomas	0.88	0.76	0.82	0.10
Object beer	0.84	0.97	0.90	0.38
Object charger	0.94	0.48	0.63	0.00
Object flying	0.92	0.59	0.72	0.00
Object hacker	0.91	0.91	0.91	0.00
Object horse	0.97	0.72	0.83	0.00
Object Lego	0.85	0.65	0.74	0.00
Object troll	0.77	0.90	0.83	0.41
Object vase	0.92	0.87	0.89	0.01
Average	0.88	0.74	0.79	0.08

Table 13 - Selecting points by contrast helps some, but nothing dramatic. The most interesting change is the video of the charger. This makes sense since this object has no texture; so only features along the edges are partly stable. See 'TLD results' for videos of using the contrast features.

WHERE DOES IT FAIL

If we look at table 11 we see that the difficult part is initializing when the object is seen in a different angle or light than during the initial image. Light and angle in themselves handled fine, but when having to initialize in these circumstances, the TLD framework fails. We saw during experiments with the tracker that it could handle rotation and illumination changes well in many circumstances. The classifier, on the other hand, could not handle these changes. For the classifier to be able to recognize the object in a new light or from a new angle, it has to learn. The TLD framework should create these training samples, but seemingly they have not been added - or at least not satisfactory. We can get a feel for what is happening by looking closer at model growth and see which patches are added. Everything has been fine up to the points of "Hide-angle" where the eye disappears and reappears seen from another angle. This angle has been seen earlier and should thus – ideally – not be a problem. The same can be said for disappearing and reappearing in another light. Ideally the framework has learned during rotation of the head and slow illumination change. It turns out that the patches showing the object in new light and angle is not always added to the model. This is because the classifier detects the patch a slightly different position than the tracker. A simple fix is to grow when a nearby patch is detected as well. Doing so lead to the desired growth of the relevant patches. But surprisingly performance drops when adding more patches, even though all additions are correct. To understand this I will start with the question "does adding more positive samples improve performance?".

Perfect growth

So far I have considered adding correct patches to the model as the road to success, but this might not be. Adding all ground truth patches to the model before start should test this.

Setup and results: All ground truth patches are added to the model before analysis. Thereafter all remains the same, the TLD framework initialize, tracks, grows and prunes as usual.

Though performance is good, it is far from perfect. Recall is typically better, while precision is lower. The average result remains almost the same as the standard version. This points out a problem in the framework: even with a perfect object model there are many errors and the amount of false positives rise.

Perfect model / Standard									
Challenge	P			R			F		
Move (S)	1.00	/	1.00	1.00	/	1.00	1.00	/	1.00
Move (L)	1.00	/	1.00	1.00	/	1.00	1.00	/	1.00
Closed	1.00	/	0.93	1.00	/	0.93	1.00	/	0.93
Gaze	0.95	/	**0.99**	0.95	/	**0.99**	0.95	/	**0.99**
Blur	0.97	/	**0.99**	0.95	/	0.95	0.96	/	**0.97**
Rotation	**0.87**	/	0.83	**0.87**	/	0.83	**0.87**	/	0.83
Angle	**0.93**	/	0.90	**0.93**	/	0.90	**0.93**	/	0.90
Light	1.00	/	1.00	1.00	/	1.00	1.00	/	1.00
Occlusion	0.95	/	**0.96**	**0.94**	/	0.93	0.94	/	0.94
Disappearance	0.75	/	**0.99**	**0.91**	/	0.87	0.81	/	**0.92**
Hide - angle	0.67	/	**0.96**	**0.89**	/	0.49	**0.76**	/	0.65
Hide - light	0.55	/	**0.88**	**0.74**	/	0.67	0.63	/	**0.76**
Clutter	0.45	/	**0.76**	**0.71**	/	0.66	0.55	/	**0.70**
Average	0.85	/	**0.94**	**0.92**	/	0.86	0.88	/	**0.89**

Table 14 – Even with perfect growth the framework still has many false positives and negatives.

How can we understand this? Over fitting is often an issue, but I don not believe that to be the explanation in this case. The amount the tree generalizes is predetermined and does not change by excessive training. At some point, new patches will simply be ignored, as they are identical to existing branches. Rather it is an issue of having to many branches compared to the forest and to low a threshold. In cases, where the object is not in view, another patch is often identified, leading to a fall in precision. Each branch on each tree represents a view of the object. And the forest believes the patch to be the object if the majority of the trees do. The reason we have several trees to begin with is for them to even each other out because they all fail at times. But we have no way of detecting whether one tree believe that the patch is the object seen from the front, while another votes for the object being seen in profile etc. The forest might, in other words, vote for an inconsistent transformation and the more branches the larger is this risk. The recall rate shows us that we cannot simply raise the threshold to fix the problem. If we increase the resolution of the scanner, we can avoid the issue at the cost of speed. Modifying the classifier seems to be the most promising strategy.

CONCLUSION

We are getting to the end. The challenges involved in object recognition have been discussed and a long-term tracking framework has been introduced and tested on eye tracking. What have been learned and where to go from here?

SUMMARY

The goal has been to create a long-term tracker that works from a single ostensive definition. Any long-term tracker is faced with the problems of change in light, object pose, object appearance, occlusion, angle and much more. To create a long-term tracker we need some way of detecting the object. This is often solved using a classifier that has been trained with large amounts of samples. Gathering these samples can be difficult – they need to be representative of possible object appearances as well as including relevant negative samples. Manually creating these can be time consuming as well as difficult. Semi-supervised learning can in some cases be a solution. One way to create a semi-supervised classifier is to use the spatial- and temporal constraints present in videos. The TLD long-term tracker uses these structural constraints to create a classifier and tracker based on a single sample in a video. The resulting tracker can track, learn and detect the object in real-time. The TLD framework and components has been tested on analyzed.

During testing of the median flow tracking approach we have seen that it works well in many scenarios. Its adaptive nature, however, makes it vulnerable to occlusion and by design it handles out of plane rotation and scaling poorly. If points are not evenly dispersed it will glide off the object. We have seen this in practice as well as understood it theoretically. In general the tracker works best with highly patterned objects and sub patches. And due to the tracker insisting on tracking by an adaptive boundary box, it has difficulties when the original patch cannot be captured by the initial box and ratio. The classifier too has difficulties with occlusion and is very sensitive to angle and light changes as well as other minor transformations. This means that many patches need to be grown if the classifier is to be useful.

The proposed TLD framework performs better in test than its components individually, but the resulting classifier still has weaknesses. The framework can recover from occlusion, but it does not necessarily detect occlusion while it happens. And recognizing an occluded object is still not possible. The problem of occlusion cannot be solved within the current framework (unless it is the same occluding object every time and hence added to the model). More problematic is it that learning light and angle variation is still troublesome. We can track through these changes, but learning to initialize under drastically changed settings is difficult and progression is slow. We thus see that the TLD tracker frequently fail when the object is lost and reappears in another light. Adding failure detection to the median flow tracker greatly helps on precision while leaving the recall rate the same. This indicates that the failure detection

has a high degree of precision, making it a valuable modification of the raw median flow tracker.

	Eyes	Eye segments	Objects
Tracker	0.16	(0.60)	0.37
Classifier	0.36	0.63	0.69
TLD	0.65	0.81	0.83
TLD-modified	0.80	0.89	0.84

Table 15 - The 3 different test domains and their F-values. The results show the performance of the tracker, classifier, TLD and the modified TLD. Notice that the pure tracking results for eye segments is initialized in each segment. See 'TLD results' for videos of the final long-term tracker.

Looking at the eye videos we see that the framework indeed boost performance. The raw Median flow tracker has an f-value of 0.16, a classifier that has been grown with affine transformations on the initial frame has 0.36f and the modified TLD framework has 0.80f. As such the framework is a success, but no reason to stop here. Before discussing modifications and improvements, a brief note on the thesis process.

A NOTE ON PROCESS AND PRODUCTION

I have not contacted or otherwise obtained any of the authors' source code, but chosen a more explorative approach. There are both benefits and weaknesses to this strategy. In the context of learning I have found it rewarding to implement the algorithms myself based solely on the articles and make my own modifications and solutions where the articles left explanatory gaps. Further this approach has the potential to generate another understanding and other ideas. On the other hand building on already tried-and-tested algorithms provides a solid foundation for performance testing and further development, both with respect to quality and speed. Obtaining the source codes is a natural next step, to compare and learn from already explored and tested solutions.

DISCUSSION

It has been demonstrated that the TLD combination of tracker and classifier outperforms both taken individually. Increasing the quality of tracker and/or classifier will further increase the quality of the framework. Any tracker that represent by a bounding box can replace the Median flow tracker. And any classifier that allows for incremental growth can replace the sequential forest. This allows for many variations and potential improvements. But already with the current – somewhat flawed – tracker and classifier, do the framework work well.

We have seen that the tracker is a surface tracker, making it ill adapted for cases where we are interested in the object as a unit. We have learned that only objects with a rich pattern can be confidently tracked and classified; this rules our many objects (e.g. most chairs). This is not an issue when tracking eyes, but if it is to be used elsewhere this could become a problem. On the other hand will objects with a constantly changing pattern also lead to failure – either by excessive growth or none (e.g. a television). It has further been demonstrated that the Median flow tracker encounter problems during scaling and rotation if confronted with an uneven point distribution. This problem has been established in theory as well as in practice. It could therefore be an idea to replace the tracker with a solution that can handle scaling and rotation more confidently. The current point filtering consists in only using the 50% most confidently tracked points. We can use Ransac to eliminate outliers and the current need to do subsequent failure detection by ensuring that the suggested transformation is consistent with the points.

Looking at the videos used for testing we see that we are indeed capable of tracking eyes (and other objects) through many and strong transformations with decent accuracy, in real–time and all from a single sample. The model grows – with correct samples – and these are used to recuperate from tracking failures. Further the system slowly learns how the object might also appear. But there is still plenty of room for improvements.

Illumination issues

In the performed tests on eyes, subjects introduced some of the light changes slowly and afterwards returned to their initial position. This is the ideal user, but not a realistic scenario. Often a user will not return to the same light and angle if they are allowed to move freely. When this loop is missing the classifier will not learn, or rather: it will learn in a slower pace. The sensitivity to large changes in light (and clutter) is not an issue if the camera is stationary – at least not to the same degree. This makes the system most appropriate for mounted cameras such as a desktop computer, surveillance camera or other statically mounted cameras. The current version is in other words less appropriate for handheld devices. I see two solutions to this: grow more patches or use a more illumination insensitive classifier. More growth can be created by:

- Putting greater trust in our tracker and grow all tracked patches. This would require that we knew with great certainty when to trust our tracker (and classifier).
- Loosening the criteria for agreement between classifier and tracker. In the current setup they need to agree completely. Instead we can accept a trajectory for growing when there is a large degree of overlap between the two.

Growth issues

Before adding any more patches to the forest and model we should confront the most blatant problem – namely that the framework, even with a perfectly pre-grown model, has errors. If we use a high resolution scanning and only use the classifier we get perfect results, but this is not an option due to the processing time. The problem is a classifier problem. Adding many patches, though they are correct, leads to many false positives. To solve this a modification of the classifier is necessary:

- A reason for the failures might be the binary character of the trees, as they potentially lead to growing and pruning of the same branch over and over again. To avoid this, the trees could be made to output a probability instead. Pruning would thus not remove a branch but simply alter its likelihood.
- Including negative samples in the model would allow a nearest neighbor search once the forest approved the patch.
- And finally it seems relevant to ensure that the individual trees are voting for the same transformation. The crude way to do this is to only accept a patch if the same growing event has caused all the approved branches. This is probably overly harsh, as we would like to accept minor differences. More interesting one could use the tracker to create pose estimates and add these estimations to the model to ensure consistent transformations when detecting.
- Adding yet another filter beside the forest and template matching is also a possibility. So far colors have not been used. Including histogram matching as yet another filter works to filter out false positives.
- Filtering by image quality: When growing a trajectory, all intermediate points are added to the model independent on their quality. This leads to a noisy model with badly illuminated and blurred samples. Ideally these samples are identified and avoided or otherwise registered as being insecure at best. Conversely many of the false positives are detections in seemingly uniform areas such as shadows or on a wall. In these cases it is simply random noise patterns that happen to be accepted. Ideally our classifier rejects both.
- Context: False positives are bound to happen. And when dealing with eyes where left and right look similar (not to mentioned other person' eyes), using context is an obvious way of differentiating. Grabner et al. has shown that the use of supporters can create a robust tracker that can handle total occlusion (Grabner, Matas, Gool, & Cattin, 2010). These companions could also be used as classification validators. Once other parts of the image have been found to co-vary with the tracked patch, these can be used for tracking as well as classification.

- Keypoints: Using interest points might be more robust than the randomly chosen feature patches in the current approach; these could also be used when tracking and would allow recognition during occlusion. Further they would likely make the classifier less sensitive to light changes. But extracting these are computationally expensive and speed is an issue. FAST corner detection (Rosten & Drummond, 2006) might be a possibility, but it is unclear whether they are stable enough for our purpose.

Figure 82 - Example of added patch. Adding blurred patches will degrade object model and should be avoided.

Final words

The presented framework works well under a wide variety of circumstances, but modifications are necessary if the current system is to be robust enough for long-term eye tracking on cell phones. At this point the system can handle noise, blur, occlusion, rotations, scale and object transformations as long as light, and preferably surroundings, remains stable. Angles can also be learned if light is stable, but permanent and fast changes in light are difficult to adapt for. Much could be gained with a more robust classifier. More patches could then be grown without fear of decreasing performance. The most obvious way to create a larger model is to add trajectories, not only when the exact patch on the trajectory has been recognized, but also when nearby patches is detected. Doing so depends on a more robust classifier.

I believe that the proposed method has shown its value even in this early stage of its development. Much work needs to be done before we have anything that resembles the ease and width of which we as humans learn, but I consider learning by tracking a valuable contribution to the long list of robust methods. And as the framework is a wrapper method it is likely to benefit from general advances within classification and tracking. In the broad perspective this form of online adaptive learning has the potential to be used far and wide. The proposed method might not be Foxconn robot material yet, but its plug-and-play nature does make it a stepping-stone towards simple integration of concept acquisition in robotics, whether it be domestic or industrial.

BIBLIOGRAPHY

Özuysal, M., Fua, P., & Lepetit, V. (2007). Fast Keypoint Recognition in Ten Lines of Code. In *In Proc. IEEE Conference on Computing Vision and Pattern Recognition.* Minneapolis, Minnesota, USA: IEEE Computer Society.

Aanæs, H. a. (2012). Interesting Interest Points. *International Journal of Computer Vision , 97*, pp. 18-35.

Andriluka, M., Roth, S., & Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. In *CVPR* (pp. 1-8). IEEE Computer Society.

Biederman, I. (1987 April). Recognition-by-components: A theory of human image understanding. *Psychological Review , 94* (2), pp. 115-147.

Bouguet, J. (2000). *Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the Algorithm.* Intel Corporation, Microprocessor Research Labs. OpenCv.

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV - Computer Vision woth the OpenCV Library.* (M. Loukides, Ed.) USA: O'Reilly.

Breiman, L. (2001 1-Oktober). Random Forests. *Machine Learning* , pp. 5-32.

Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., & Gool, L. J. (2009). Robust tracking-by-detection using a detector confidence particle filter. In *ICCV* (pp. 1515-1522). IEEE.

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence* (47), pp. 139–159.

Cerman, L., Hlavac, V., & Matas, J. (2009). Sputnik Tracker: Having a Companion Improves Robustness of the Tracker. In *Scandinavian Conference on Image Analysis (SCIA)* (pp. 291-300).

Chabris, C. a. (2010). *The Invisible Gorilla: And Other Ways Our Intuitions Deceive Us* (1 ed.). Crown Archetype.

Christoudias, C., Urtasun, R., Kapoor, A., & Darrell, T. (2009). Co-training with Noisy Perceptual Observations. In *CVPR* (pp. 2844-2851). IEEE.

Dreyfus, H. (1992). *What computers still can't do: a critique of artificial intelligence.* London, England: The MIT Press.

Dreyfus, H. (2008). Why Heideggerian AI Failed and How Fixing It Would Require Making It More Heideggerian. In P. Husbands, O. Holland, & M. Wheeler (Eds.), *The Mechanical Mind in History* (pp. 331–371). The MIT Press.

Godec, M., Leistner, C., Bischof, H., Starzacher, A., & Rinner, B. (2010). Audio-Visual Co-Training for Vehicle Classification. In *Proceedings of the 2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance* (pp. 586-592). Washington, DC, USA: IEEE Computer Society.

Grabner, H., & Bischof, H. (2006). On-line Boosting and Vision. *CVPR*, *1*, pp. 260-267.

Grabner, H., Matas, J., Gool, L. J., & Cattin, P. C. (2010). Tracking the Invisible: Learning Where the Object Might be. In *CVPR* (pp. 1285-1292). IEEE.

Hapgood, F. (2006 15-December). Factories of the Future. *CIO , 10* (6), p. 118.

Harris, C. a. (1988). A combined corner and edge detector. *Proceedings of the 4th Alvey Vision Conference* , pp. 147–151.

Haugeland, J. (1985). *Artificial Intelligence: The Very Idea.* Cambridge, Mass, US: The MIT Press.

Kalal, Z., Matas, J., & Mikolajczyk, K. (2009). Online learning of robust object detectors during unstable tracking. In *Proceedings of the IEEE On-line Learning for Computer Vision Workshop (2009)* (pp. 1417-1424). Kyoto, Japan: On-line Learning for Computer Vision Workshop.

Kalal, Z., Matas, J., & Mikolajczyk, K. (2010). P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints. San Francisco, CA: Conference on Computer Vision and Pattern Recognition.

Kalal, Z., Mikolajczyk, K., & Matas, J. (2010 йил 23-August). Forward-Backward Error: Automatic Detection of Tracking Failures. *Pattern Recognition (ICPR), 2010 20th International Conference on* , pp. 2756-2759.

Lepetit, V., Lagger, P., & Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *CVPR* (Vol. 2, pp. 775-781). IEEE.

Lucas, B. a. (1981). An iterative image registration technique with an application to stereo vision. *IJCAI*, *2*, pp. 674–679.

Matas, J., & Vojir, T. (2011). Robustifying the Flock of Trackers. In S. S. Andreas Wendel (Ed.), *16th Computer Vision Winter Workshop.* Mitterberg.

Millikan, R. G. (2000). *On Clear and Confused Ideas: An Essay about Substance Concepts.* Cambridge University Press.

Mundy, J. (2006). Object Recognition in the Geometric Era: a Retrospective. In J. Ponce, M. Hebert, C. Schmid, & A. Zisserman (Eds.), *Toward Category Level Object Recognition, volume 4170 of Lecture Notes in Computer Science* (pp. 3-29). Springer.

Perera, D. (2008 7-July). *Defense Systems.* Retrieved 2012 йил 20-4 from http://defensesystems.com/articles/2008/07/neural-nets-find-niche.aspx

Ramanan, D., Forsyth, D. A., & Zisserman, A. (2005). Strike a pose: tracking people by finding stylized poses. In *CVPR* (Vol. 1, pp. 271-278). IEEE.

Rosenberg, C., Hebert, M., & Schneiderman, H. (2005). Semi-Supervised Self-Training of Object Detection Models. In *Seventh IEEE Workshop on Applications of Computer Vision* (Vol. 1, pp. 29-36). Washington, DC, USA.

Rosten, E., & Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision* (s. 430-443). Springer.

Taylor, S., Rosten, E., & Drummond, T. (2009). Robust feature matching in 2.3µs. In *CVPR* (pp. 15-22). Miami, USA: IEEE.

Trucco, E., & Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision.* Prentice Hall.

Viola, P. a. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Vol. 1, pp. 511-518). Hawaii, USA.

Yilmaz, A. J. (2006 December). Object Tracking: A Survey. *ACM Computing Surveys , 38* (4), p. 45.

Zhou, Y., & Goldman, S. (2004). Democratic Co-Learning. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence* (pp. 594 - 602). Washington, DC, USA: IEEE Computer Society.

Zhou, Z.-H., & Li, M. (2005 November). Tri-Training: Exploiting Unlabeled Data Using Three Classifiers. *IEEE Trans. on Knowl. and Data Eng. , 17* (11), pp. 1529-1541.

Zhu, X. (2008 19-July). *Semi-Supervised Learning Literature Survey.* Retrieved 2011 23-November from http://pages.cs.wisc.edu/~jerryzhu/research/ssl/semireview.html