Evaluation of TLD/Predator algorithm

BALACHANDER CHOKKALINGAM

KTH Datavetenskap
och kommunikation

Degree project in
Computer Science
Second cycle
Stockholm, Sweden 2013

KTH Computer Science
and Communication

Evaluation of TLD/Predator algorithm

Evaluation of TLD(aka Predator) algorithm used for real-time tracking of unknown objects in a video stream from eye tracking perspective.

NAME BALACHANDER CHOKKALINGAM

Master's Thesis at NADA
Supervisor and Examiner: Stefan Carlsson
Supervisor at Company: Kai Hübner

TRITA xxx yyyy-nn

Acknowledgements

This thesis work is completed as a part of the curriculum towards the completion of the master's program in Systems, Control and Robotics at Kungliga Tekniska Högskolan (KTH), Sweden. This work is done at Tobii Technology AB under the guidance and supervision of Kai Hübner from April 2012 to December 2012.

To start with, I would like to thank Tobii Technology AB for giving me the opportunity to work on this degree project; it has been an incredibly rewarding experience.

I would like to take this opportunity to convey my gratitude to Kai Hübner for his consistent guidance and support throughout my work. His support was beyond the boundaries of a supervisor. His supervision helped me to amend and improvise the research and thereby making it a much better one. I specially thank him for his infinite patience.

I would like to thank Mattias Hanqvist for providing me with this opportunity, and for the support at the company. I would also like to thank everyone, who were directly or indirectly responsible for creating this thesis opportunity.

I would like to thank Stefan Carlsson for support he rendered, and for being my academic supervisor and examiner.

Last but not the least; I would like to extend my warm thanks to my friends, colleagues and parents, without whom it wouldn't have been possible to accomplish this.

On a different note, many people have been a part of my graduate education and I am highly grateful to all of them.

Abstract

TLD is an award-winning, real-time algorithm for long-term tracking of unknown objects in video streams. The object of interest is defined by a bounding box in a single frame. TLD simultaneously **t**racks the object, **L**earns its appearance and **D**etects it whenever it appears in the video. The result is a real-time tracking that typically improves over time. Long-term tracking of arbitrary objects is a the core problem in many computer vision applications: surveillance, object auto-focus, SLAM, games, HCI, video annotation etc.

The following work is done:

1. The OpenTLD algorithm is extended for Multi-Object Tracking.
2. The OpenTLD algorithm is evaluated (i.e. problems with the algorithm, usability of the algorithm), specially from Eye tracking perspective.
3. The OpenTLD algorithm is compared with other tracking algorithm e.g. Mean shift in OpenCV in terms of tracking performance.
4. Some enhancements to the OpenTLD algorithm are made.

Contents

Chapter 1

Introduction

TLD [4] is a novel tracking framework that explicitly decomposes the long-term tracking task into tracking, learning and detection. The tracker follows the object from frame to frame. The detector keeps track of all the appearances that have been observed so far and corrects the tracker if necessary. The learning estimates the detector's error and updates it to avoid these errors in the future. A novel method called P-N learning [3] is used to identify detectors errors and learn from them.

Due to its learning abilities TLD has been advertised under name *Predator*. TLD has been developed by Zdenek Kalal during his PhD thesis supervised by Kyrstian Mikolajczyk and Jiri Matas in university of Surrey.

Before TLD algorithm state-of-the art methods for object tracking performed tracking-by-detection, meaning that a detector predicts the position of an object and adapts its parameter to object's appearance at the same time. This method work well if there is no occlusions, but tends to fail in cases of occlusions. TLD(Tracking-Learning-Detection) overcomes this problem by training the detector with examples found on the trajectory of a tracker which does not depend on the detector. By separating tracking and detection high robustness is achieved and performs better that the earlier tracking-by-detection methods. The detector uses cascaded approach to speed up computing, and uses simple features for object detection.

The algorithm is extended to track multiple objects (currently only one object can be tracked by OpenTLD algorithm), the ability to track multiple object is needed mainly cause two eyes need to be tracked at the same time for eye-tracking purpose.

The OpenTLD algorithm is also changed to stream like architecture which facilitates evaluation of individual components of the algorithm and replacement of those components by other possible components.

The evaluation of the algorithm is needed to check the usability of Algorithm for

the purpose of eye tracking and to check how well can it implemented on a DSP board. The evaluation is also needed to check any problems and propose ways to overcome them. Python scripting is used to evaluate the algorithm.

The performance, frame rate and memory usage of the algorithm is analysed for variation in the parameters of the algorithm and then few enhancements to the algorithm is made. It is also checked how the algorithm performs for various sizes of the images and various sizes of the initial bounding box.

The OpenTLD algorithm is also compared with Median Flow Tracker, OpenCV meanshift detector, mean shift detector from [29] and OpenCV Eye Detector in terms of performance and frame rate. In which situation to use which algorithm is analysed and ways to combine them is proposed.

We conclude by mentioning the future scopes of the algorithm and highlighting the main aspects of the work done.

Chapter 2

Object tracking

This chapter explains aspects of object tracking algorithms and is summarization from [5] [6] [7].

The aim of object tracking is the estimate the object location in the image sequence when initial position of the object is given in the first frame. Additionally based on the application a tracker can also calculate object based information, orientation of the object, area of the object, and/or shape of the object.

The advancement in faster computers, the easy availability of inexpensive cameras with high image quality, and increase in the need for automation video analysis has led to lot of interest in object tracking algorithms.

The main steps of video analysis are

1. Detection of object of interest with motion in the image sequence.
2. Tracking of such object from frame to frame.
3. Analysis of behaviour of object tracks.

The applications of object tracking are:

- Motion-based recognition, recognition of an object and/or it motion, based on motion in a series of images e.g. human identification based on the gait.
- Automated surveillance, scene monitoring to detect suspicious activities or unlikely events.
- Video indexing, automatic annotation and retrieval of the videos in multimedia databases.
- Human-computer interaction, recognition of gesture, tracking the eye gaze for data input to computers, etc.
- Road traffic monitoring, real time gathering of traffic statistics to direct traffic

flow.

- Vehicle navigation, i.e. video-based path planning and obstacle avoidance capabilities.

Object tracking has remained a **challenging task** even after decades of research as a result of factors such as

- Illumination change.
- Pose variation.
- Scale variation.
- Object deformation.
- Motion blur, noise in video/image sequence.
- Partial/full occlusion of the object.
- Loss of information due to projection of 3D on 2D image.
- Real-time processing requirements.

However, the tracking can be simplified by imposing constraints on the motion and/or appearance of the objects. Most of the tracking algorithms assume that smooth object motion with no abrupt changes. The object motion can be further constrained to be constant velocity or constant acceleration based on a priori information. The problem of object can be further simplified by using the prior knowledge about the number and the size of objects, or the object appearance and shape.

The approaches taken to solve the problem have been numerous. They differ from each other based on the way they approach the following questions

1. Which object representation is suitable for tracking?
2. Which image features should be used?
3. How should the motion, appearance, and the shape of the object be modelled?

The answers to these questions depend on the context/environment in which tracking is performed and the end use for which the tracking information is being sought. Many of the tracking methods have attempted to answer these questions for a variety of scenarios.

2.1 Object Representation

In an object tracking scenario, an object can be defined as anything that is of interest for further analysis. Some of the examples of the object are vehicles on a

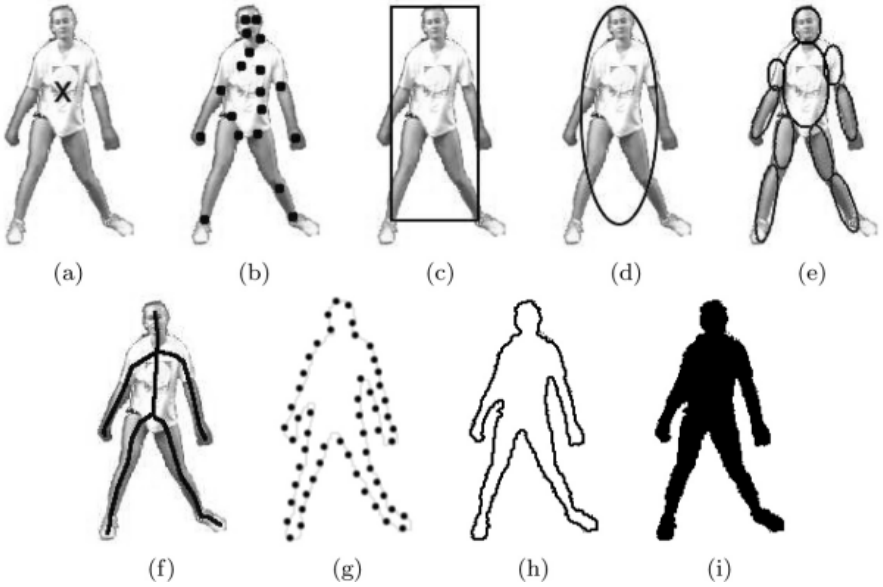

Figure 2.1. Object representations. (a) Centroid, (b) multiple points, (c) rectangular patch, (d) elliptical patch, (e) part-based multiple patches, (f) object skeleton, (g) complete object contour, (h) control points on object contour, (i) object silhouette. Image is from [7] .

road, people, planes in air, eyes, faces, animals in forest, marine species in sea etc. Objects can be represented by their shapes and appearances. Refer to Figure 2.1 for various object representations.

Some of the commonly used **object shape representations** are:

- **Points**: The object can be represented by a centroid or by a set of points. Point representation is more suitable when the tracking object is small.
- **Primitive geometric shapes**: Object shape is represented by a rectangle, ellipse. Though primitive geometric shapes are more suitable for representing simple rigid objects, they are also used for tracking non-rigid objects.
- **Object Silhouette and contour**: Contour representation defines the boundary of an object. The region inside the contour is called the silhouette of the object. Silhouette and contour representations are suitable for tracking complex non-rigid shapes.
- **Articulated shape models**: Articulated objects are made of component that are connected together For example, the human body is an articulated object with torso, legs, hands, head, and feet connected by joints.
- **Skeletal models**: Object skeleton can be extracted by applying medial axis transform to object silhouette [8]. For example, humans could be represented using human skeletons.

Also below is some **common appearance representation** used in object tracking:

- **Probability densities** of object appearance. The probability density estimates of the object appearance can either be parametric, such as Gaussian and a mixture of Gaussian's or non-parametric such as histogram.
- **Templates** are formed using simple geometric shapes or silhouettes. An advantage of a template is that it carries both spatial and appearance information.
- **Active appearance models** consist of statistical model of object shape and appearance.
- **Multi-view appearance models**: In this methods appearance models for different views of the object is associated with different directions.

The relationship between the object representations and the tracking algorithm is generally strong. The object representations are usually chosen according to the application domain.

2.2 Feature Descriptors

The choice of correct features is crucial to tracking. In general, the most desirable property of a visual feature is its uniqueness so that the objects can be easily distinguished in the feature space. Choice of feature is closely related to the object representation. For example, object edges are usually used as features for contour-based representation, and colour is used as feature for histogram-based appearance representations. The invention of different image features that capture different characteristic properties is motivated for difficultly in object detection due to varying object appearance due to many factors such as viewpoint, occlusion, illumination, texture and articulation.

Some of the methods base their detectors on single type of features, while others integrate multiple features. Below we discuss the different types of **feature descriptors** used for representation of the object:

- **Gradient features**: one main category of gradient based methods is to use shape/contour to represent objects such as human body. Other category is to use statistical summarization, example are SIFT descriptor, SURF descriptor, HOG descriptor, Adaptive Contour Feature (ACF) etc.
- **Colour features**: Intensity-based descriptors have been widely used for feature representation of salient points. Colour descriptors are robust against certain photometric changes and have been used to increase the discriminative power. Two physical factors mainly influence the apparent colour of an object, Spectral power distribution of the illuminant and the surface reflectance

properties of the object.

- **Texture features**: Texture is a measure of intensity variation of a surface which quantifies properties as smoothness and regularity. Example: Gabor Wavelet.
- **Spatio-temporal features**: Local space-time features capture characteristic salient and motion patterns in video and provide relatively independent representation of events with respect to their spatio-temporal shifts and scales as well as background clutter and multiple motions in the scene.
- **Multiple features fusion**: Improved performance has been shown by using various combinations of existing features.

2.3 Online Learning

It object tracking, it is very important that the algorithm handles the appearance variation. There are two types of appearance variations: intrinsic and extrinsic. Examples of intrinsic variations are due to pose variation and shape deformations. Examples of Extrinsic variations are due to different illumination, camera motion, camera viewpoint, and occlusion. These variations can only be handled by adaptive methods that incrementally update their representation.

Online algorithms are divided into below categories:

- **Generative tracking methods** learn a model to represent the appearance of the object. Examples of Generative tracking algorithms are Eigen Tracking [9], WSL Tracking [10] and IVT [11].
- **Discriminative tracking methods** aim to find a decision boundary that can best separate the Object from the background. Example Avidan [12] used an adaptive ensemble of classifiers for visual tracking.
- **Hybrid methods**: Combines Generative and Discriminative methods.

Chapter 3

Gaze Tracking and Tracking Eye as an object

In this chapter we discuss about Gaze Tracking, Eye Tracker Types, information from eye tracking, and applications of Eye Tracking summarized from [13] [14]. We also discuss the gain from tracking eye as an object in Section 3.4.

Gaze tracking is the process of measuring the point of gaze or the movement of eye with reference to the head. The device used to measuring eye positions and eye movement is called eye tracker. Eye tracking is a term often alternatively used to describe gaze tracking. We use the term tracking eye as an object separately from eye tracking.

In the fundamental form of gaze tracking a person looks at another person's eye and estimates where the other person is looking. With recent innovation of highly accurate eye tracking technology, it is being widely used both to understand the human behaviour and to enhance computer interaction.

3.1 Eye Tracker Types

There are several ways to measure the rotation of the eye, but there are 4 **principal categories of eye trackers**:

1. Using a special contact lens with an embedded mirror or magnetic field sensor, and measuring the movement of the attachment assuming that it does not slip significantly as the eye rotates. This type of measurement provides extremely sensitive recording of the eye movements.
2. Measuring the electric potential of the electrodes placed around the eyes. In this method eye rotation can be detected in total darkness and also if the eyes

are closed.

3. Light, typically infra red is reflected from the eye and sensed by a video camera or some other specially designed optical sensor. The information is then analysed to extract eye rotation from changes in the reflections. This is a non-contact method unlike previous methods. Most of the eye trackers are based on the fundamental principle of corneal-reflection tracking as shown in Figure 3.1

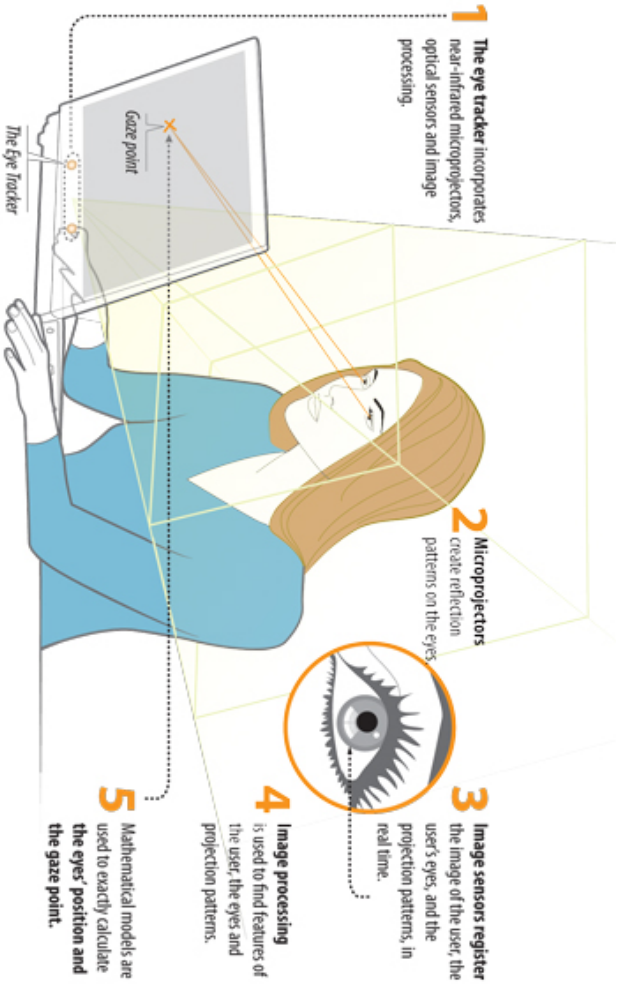

Figure 3.1. Eye Tracking Step by Step. Image is from [13] .

4. Webcam or other mobile cam can also be used for some of functionalities of an eye tracker. Not as accurate as standard eye trackers, but provide general and cost effective solutions. There is a limitation to this method by the ambient

light in the environment.

3.2 What can be tracked with respect to eyes?

- **Gaze direction and gaze point** is used in gaze interaction with computers and other interfaces, and in behavioural research/human response testing.
- **Eye presence detection** is the fundamental part of eye tracking. It can be used for applications such as power saving by dimming the screen when the eyes are not present etc.
- **Eye position** - the ability the get the position of the eyes in real time is part of what make the eye tracking system accurate and precise, while allowing the user to move freely. Eye position tracking can be used to render 3D images to the user without the use of glasses.
- **Eye identification** - Geometrical eye features and iris identification can also be used for user identification, driver identification, and automatic logon to your home PC.
- **Eyelid closure** can be used to monitor people's attention or sleepiness, for instance in advanced driver assistance or operator safety solutions.
- **Eye movement and patterns** are studied to understand human behaviour and to assess and diagnose injuries or diseases.
- **Pupil size and pupil dilation** are reliable markers of impairment, concussion, drug or alcohol influence, or emotions. Can be used in, market research, scientific research and medical assessment.

3.3 Applications of eye tracking

The eye tracking techniques are used by a wide variety of disciplines.

- **Cognitive science**, e.g. Eye movement in reading, music etc.
- **Psychology** (notably psycholinguistics, the visual paradigm), e.g. Recently eye tracking has been used to study online language processing [15].
- **Human computer interaction** (HCI), e.g. use of eyes as pointing device, usage in gaming.
- **Marketing research** e.g. Visual attention and direction of interest of website users.
- **Medical research** (neurological diagnosis): Our eyes directly reflect brain

activity and cognition. A lot can be revealed about the human brain and behaviour by studying eye movements and gaze patterns.

3.4 How object tracking algorithm can be used in eye tracking

Object tracking algorithm can be used over web-cam or other mobile cam technology to track eye as an object.

The **information's** that can be **obtained by tracking eye as an object** are:

- **Eye Presence detection**: One the tracking algorithm is initiated, no output from the tracking algorithm means that eyes that were being tracked do not exist in the scene.
- **Eye Positions**: The x, y coordinate of the eyes along with the size of the eyes tracked. One application of knowing eye positions to render 3D images without the glasses.
- **Extracted Eye**: Once the eye position is known along with the size, the eye can be extracted for further processing. This step is pre-step to gaze tracking and other information extractions such as pupil size, eyelid closure, eye identification etc.

Chapter 4

OpenTLD

This chapter gives the summary of working of OpenTLD as described in [16].

Previous to TLD state of the art methods for object tracking perform adaptive tracking-by-detection, meaning that a detector predicts the position of an object and adapts its parameters to the object's appearance at the same time. These methods were suitable when the object does not disappear from the scene, but tend to fail when faced with occlusions. TLD (Tracking-Learning-Detection) paradigm overcomes this problem. In this method a detector is trained with example found on the trajectory a tracker that itself does not depend on the object detector. By separating tracking and detection TLD algorithm outperforms existing adaptive tracking-by-detection methods. Also a considerable reduction in the computing time is achieved by using simple features for object detection and by using cascaded approach.

The block diagram of TLD frame work is shown in Figure 4.1.

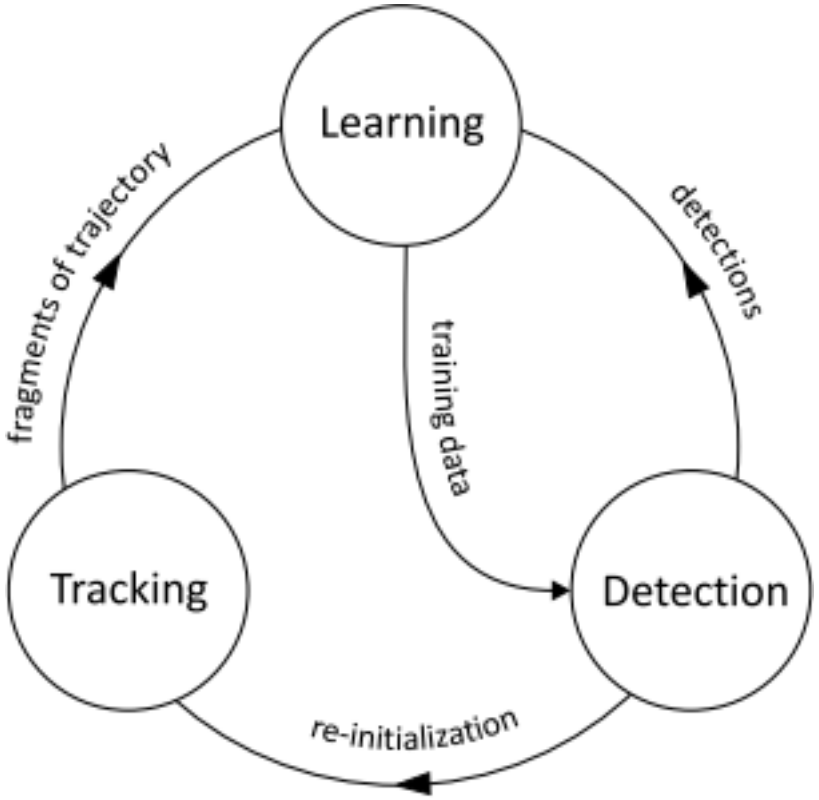

Figure 4.1. The block diagram of the TLD framework. The image is from [4] .

The TLD framework decomposes the long-term tacking task into three sub-task: tracking, learning and detection. Each sub-task is addressed by a single component and the components operate simultaneously. The tracker follows the object from frame to frame. The detector localizes all the appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detector's errors and updates it to avoid these errors in the future.

The work-flow of OpenTLD is given in Figure 4.2.

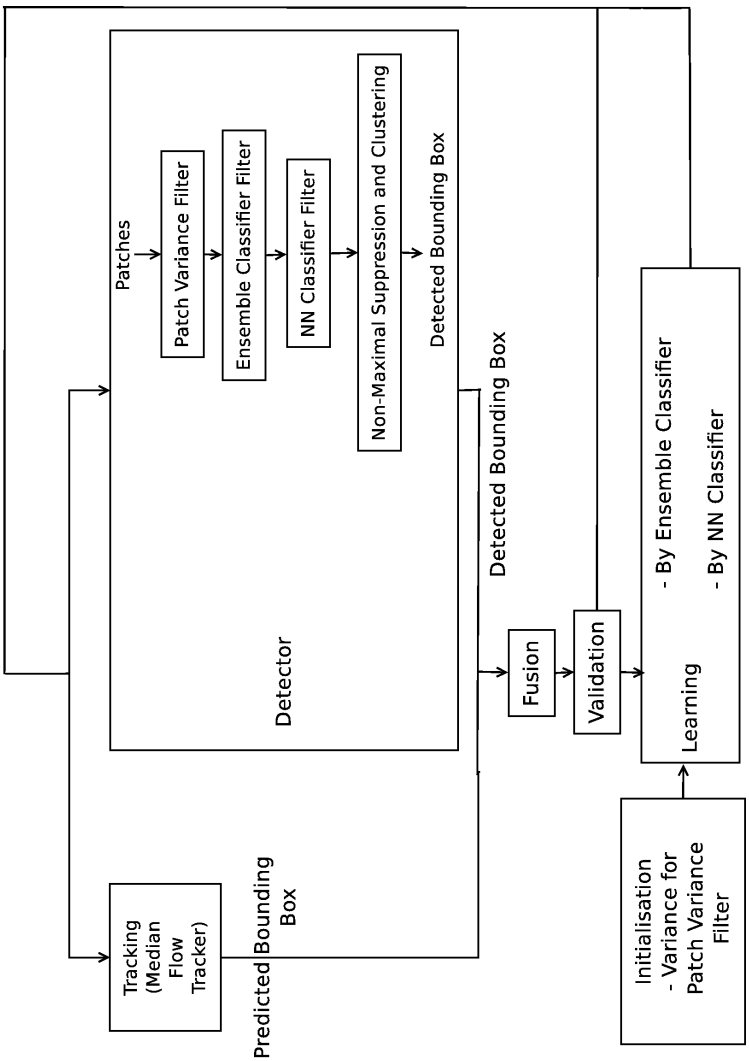

Figure 4.2. OpenTLD architecture.

The initialisation leads to a learning step. Then the tracker and detector run in parallel and their result is combined into a single final result. If this result passes a validation stage, learning is performed.

In Section 4.1, the tracker is described, then in Section 4.2, the detector cascade is explained and finally in Section 4.3 we describe how the results of the tracker and detector are combined and what happens during learning step.

4.1 Tracking

In this section we describe tracker used for recursive object tracking. The only information required by this method is the location of the object in the previous frame.

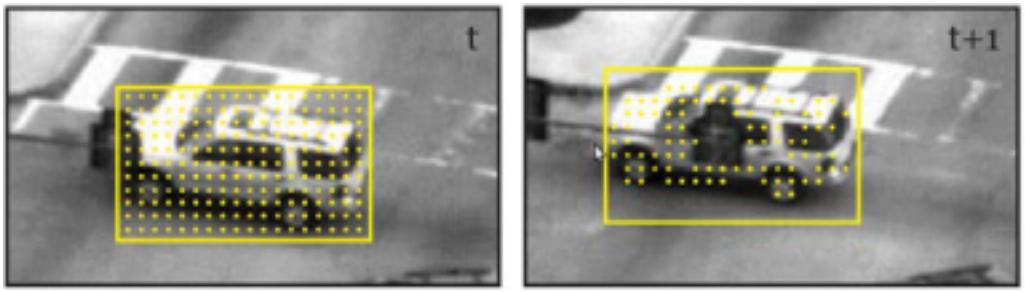

Figure 4.3. The principle of the recursive tracking method consists of tracking points using an estimation of the optical flow, retaining only correctly tracked points and estimating the transformation of the bounding box. Image is from [19] .

The method for recursive tracking is explained according to Figure 4.3. First, an equally spaced set of points in constructed in the bounding box in frame t as shown in the left image. Using Lucas and Kanade [17] optical flow is estimated for each of these points. This method works well for corner points [18] and is unable to track points on homogeneous regions. The information from the Lucas-Kanade method is used along with normalised correlation coefficient and forward backward error to filter out tracked points that are likely to be erroneous. In the right image the remaining points are shown. If the median of all forward-back error measures is below a certain threshold, then the remaining points are used to get the new bounding box in the second frame else it's considered to be an indication of the drift. A transformation model as explained in [16] is used to calculate the new bounding box in the next frame.

4.2 Detection

In this section the method for detecting the object is discussed. Object detection enables the re-initialisation of the recursive tracker that itself does not maintain an object model is therefore unable to recover from failure. The object detection mechanism does an exhaustive search in the image to find the object. Since several thousands of the subwindows are checked, most of the time of the algorithm is spent on the object detection.

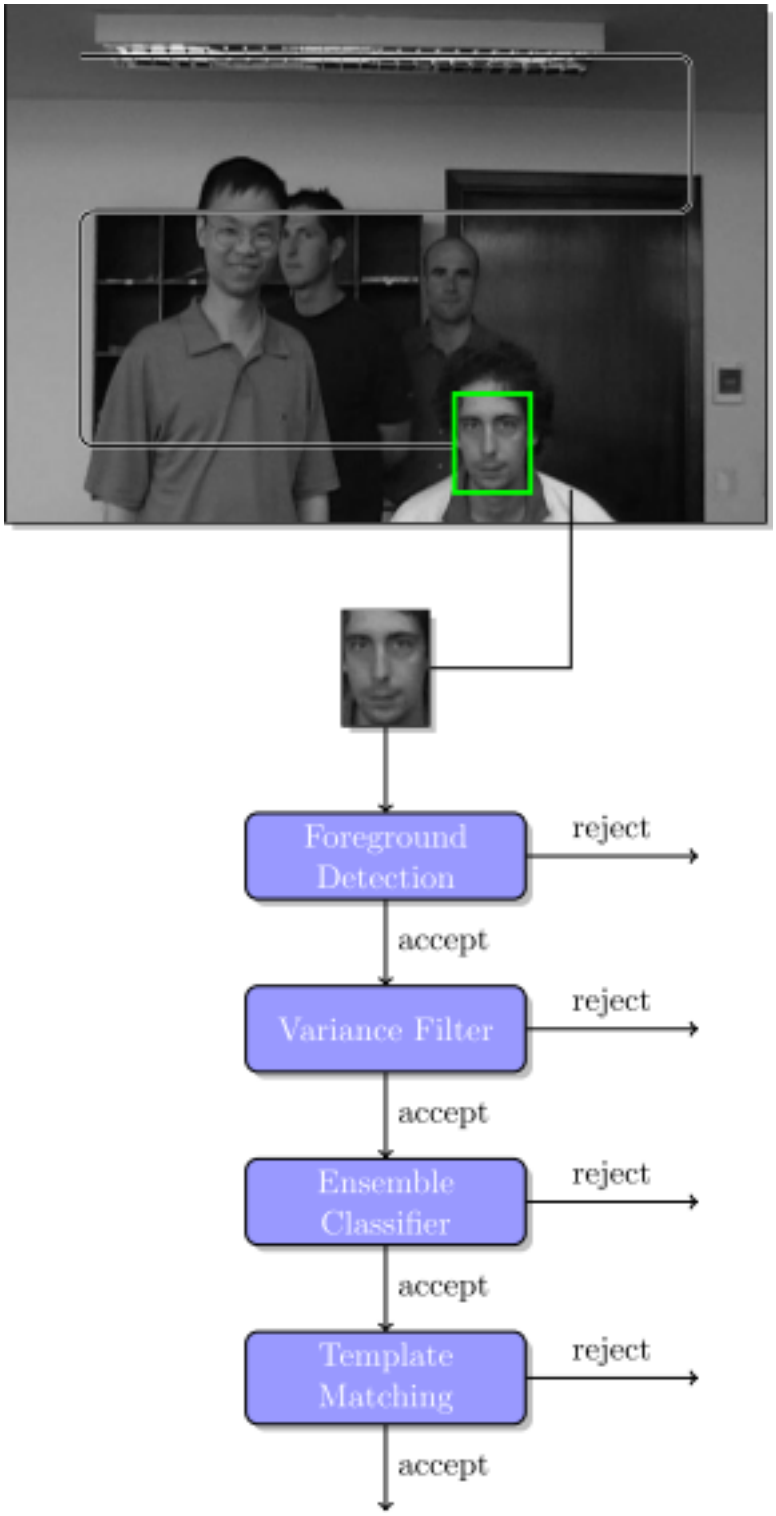

Figure 4.4. Sliding Window Approach. Cascaded approach is used in order to reduce computing time. Image is from [16] .

The object detector is based on a sliding-window approach [20] [21] as illustrated in Fig 4.4. The image is top is passed through detector cascade. The first classifier in the detector cascade goes through all the subwindows, and passes a subset of them to the next classifier in the detector cascade and so on. The four stages of the detector cascade in OpenTLD are shown below the input image. The objective of detector cascade is to filter out as many subwindows, which are not the candidate for predicted object with minimal amount of computation [22]. The first stage of the detector cascade foreground detector needs a background model. In the evaluation we do not use the foreground detector and is ignored. In the second stage variance filter all the subwindows are rejected that exhibit a variance lower than a certain threshold (by default its half the variance of initial bounding box). The next stage is the ensemble classifier based on random ferns [23]. And the final stage is based on Nearest Neighbour Classification and template matching based on NCC value as a similarity measure. If there are overlapping positive subwindows at the end of detector cascade then non-maximal suppression strategy is used to come up with the output bounding box.

The section is organized as follow. In Section 4.2.1 the Sliding Window Approach is described. In Section 4.2.2 variance filter is described. Section 4.2.3 describes the ensemble classifier. Section 4.2.4 describer the NN Classifier and Template Matching. Section 4.2.5 describes how the overlapping subwindows are combined into a single result.

4.2.1 Sliding Window Approach

In this approach for object detection, all the sub-images of the image are tested for object of interest [24] . Every possible subwindow in the image is a potential object of interest and number of such possible windows are 23,507,020,800 for an image of size 640x480 and the number of possible subwindows grows as n^4 for image size of nxn(See Appendix A.1 for the proof in [16]). The search space is restricted to

$$numWindows = \sum_{s \in 1.2^{-minScale...maxScale}} \lfloor (\frac{n - s(w + d_x)}{s d_x}) \rfloor \lfloor (\frac{m - s(h + d_y)}{s d_y}) \rfloor \tag{4.1}$$

The derivation of equation is done in [16] (See Appendix A.1 for the proof). Where $s = 1.2^a, a \in minScale...maxScale$, n = image width, m = image height, w width of initial bounding box, h = height of initial bounding box, d_x and d_y are margins between two adjacent subwindows and set to be $\frac{1}{10}$ of the values of original bounding box. Also by default the subwindow of minimum area (minSize) is of 25 pixels.

For a image size of 320×240 and bounding box size 90×39, the numWindows = 11116.

Since each subwindow is tested independently, this could be done parallely. The subwindows are tested by using as many threads as cores are available on the system.

4.2.2 Variance Filter

The Variance filter works by rejecting the image subwindows with variance less than the threshold σ^2_{min}. The variance of a image patch is the measure of its uniformity.

In Figure 4.5 regions marked with red box are examples of region with low variance, and the regions marked with green box are examples of higher variance.

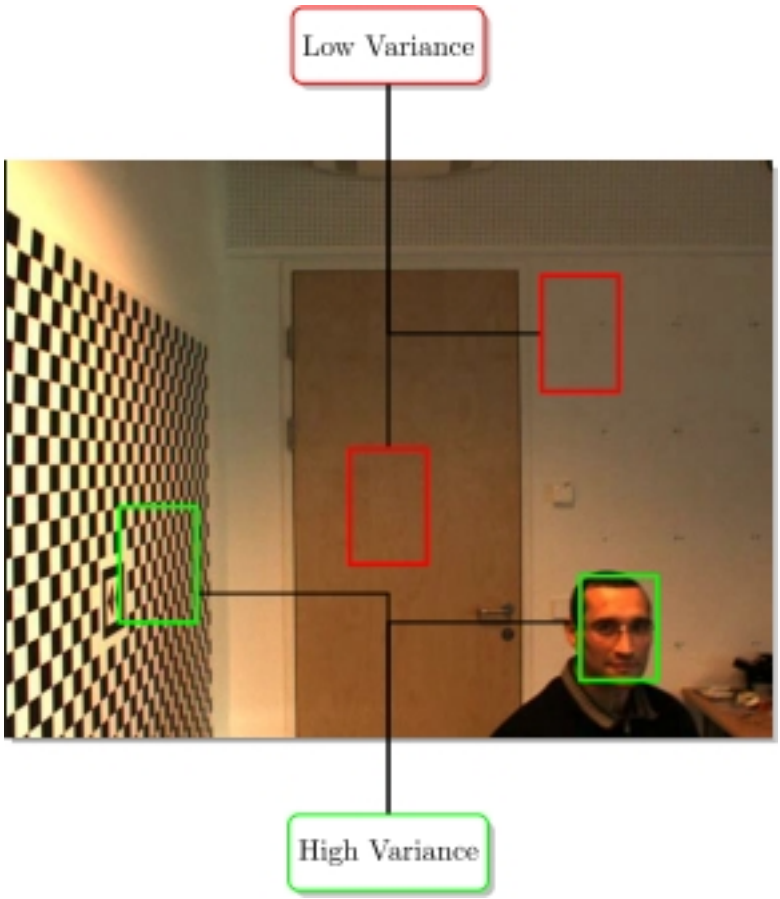

Figure 4.5. Low variance and High Variance subwindow in an image. Image is from [16] .

The means to calculate variance is made faster by using integral images as shown in equation 4.2, the derivation of the equation is shown in [16]

$$\sigma^2 = \frac{1}{n} I"(B) - [\frac{1}{n} I'(B)]^2 \tag{4.2}$$

where,

$$I'(B) = \sum_{i=1}^{n} x_i \tag{4.3}$$

, sum of pixels within a bounding box B and

$$I"(B) = \sum_{i=1}^{n} x_i^2 \tag{4.4}$$

, sum of square of pixels within a bounding box B For σ^2_{min}, by default half of the variance value of the initial selected patch is used.

4.2.3 Ensemble Classifier

In the second stage of detection cascade, ensemble classification method is employed also known as random fern classification. The classifier decision is based on a number of pair wise pixel intensity comparisons [25]. A probability P_{pos} is calculated for each of the subwindow, and is rejected for a value less than a threshold value.

The process of feature calculation is shown in Figure 4.6. In this figure, a sample image is classified. In each of the four boxes below this image a black and a white dot are shown referring to a pair of pixel in the original image.

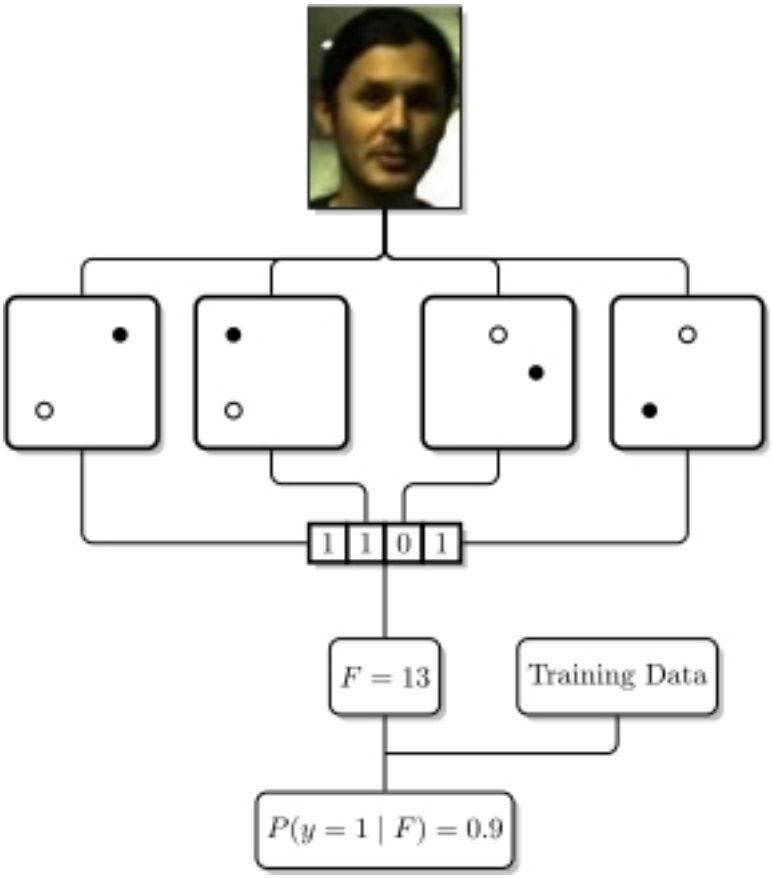

Figure 4.6. Single fern feature calculation. The i^{th} bit of the feature value is determined from i^{th} pixel pair intensity comparison. Image is from [16] .

The positions of these dots are drawn out of a uniform distribution at the start. For each of the boxes it is now tested whether in the subwindow the pixel at the position of the white dot is brighter than the pixel at the position of the black dot. This can be expressed as

$$f_i = \begin{cases} 0 & \text{if } I(d_{i,1}) < I(d_{i,2}) \\ 1 & \text{otherwise,} \end{cases}$$

where $d_{i,1}$ and $d_{i,2}$ are two random locations. This comparison is invariant against constant brightness variations.

The feature value is then used to retrieve the probability $P(y = 1|F)$ where y = 1 refers to the event that the subwindow has a positive class label. Multiple of these ferns are used and the average value is used to obtain the final confidence value P_{pos}.

4.2.4 NN Classifier and Template Matching

The final stage of the detector cascade is more restrictive but requires more computation. It uses template matching method and 1NN classification (Nearest Neighbour Classification). A list of positive and negative templates are built and used as the algorithm is learning. The learnt patches are resized to a size of 15x15 patches. For comparing two patches, the Normalised Correlation Coefficient (NCC) [16] value is used

$$ncc(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^{n} \frac{(P_1(x) - \mu_1)(P_2(x) - \mu_2)}{\sigma_1 \sigma_2} \tag{4.5}$$

where μ_1, μ_2, σ_1 and σ_2 are the means and standard deviations of patches P_1 and P_2. NCC value is closer to 1 when the two patches are similar and closer to -1 when the patches are similar. Also the distance between two patches can be calculated using the formulae

$$d(P_1, P_2) = 1 - \frac{1}{2}(ncc(P_1, P_2) + 1) \tag{4.6}$$

The distance value is between 0 and 1. The positive and negative templates are maintained in positive class as P^+ and negative class as P^-.

In Figure 4.7 the positive and negative templates are shown that were learned on the sequence *Multi Face Turning* (See [16] for description).

Figure 4.8 shows the distance d^+ and d^- of unknown image patch the closest member of positive class and closest member of the negative class. The distances are combined to express the confidence (p^+) whether the patch belongs to the positive class.

$$p^+ = \frac{d^-}{d^- + d^+} \tag{4.7}$$

A subwindow is accepted if the confidence is greater than a threshold.

(a) Positive Examples (b) Negative Examples

Figure 4.7. Positive and negative patches acquired for the template matching method during a run on a sequence from SPEVI dataset. Image is from [16] .

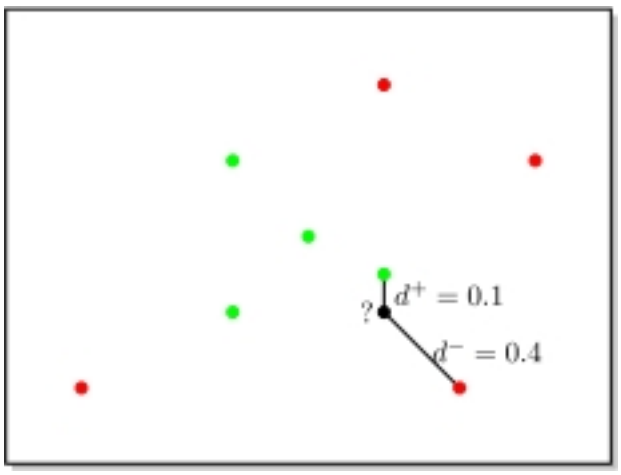

Figure 4.8. The distance of a unclassified patch labelled with ? to the positive class is d^+ and to the negative class is d^-. Image is from [16] .

4.2.5 Non-maximal Suppression and Clustering

After the detector cascade stage there are more than one subwindow that is a possible candidate for object. As shown in Figure 4.9 there are multiple subwindows with high confidence that occur around the true detection shown in green. According to Blaschko [26] it is problematic to choose the subwindow with highest confidence, Instead it is desirable to employ non-maximal suppression strategies that combine the result.

The method described in [20] that clusters detections based on their spatial overlap is used for non-maximal suppression. For each of the cluster, the bounding boxes are average and combined into a single result.

Then a hierarchical clustering algorithm described in [27] is used that works a follow. First the pair wise overlap between all the confident bounding box is calculated. Then starting from one bounding box the nearest bounding box is checked. If the distance is lower than a certain threshold, they are put together in a single cluster and if one of this bounding was already in a cluster, the clusters are merged. If the overlap is greater than the cut-off threshold, they are put into a different cluster. This is continued for remaining bounding box.

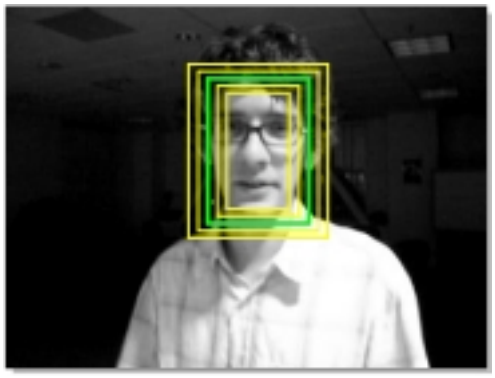

Figure 4.9. Overlapping subwindows with high confidence (yellow) are averaged to get a single result (green). Image is from [16] .

4.3 Learning

The tracker estimates the position of the object in the new frame and detector detects the object in the new frame. In this section we describe how their results are combined and also what happens during the learning step and when it's performed.

There is no learning process in the variance filter during the algorithm run, while the Ensemble Classifier and NN Classifier based on template matching are trained online. Also a criteria is defined to address the template update problem in order to consider the final result suitable for performing a learning step. In learning PN constraints are enforced [3]. The P constraints require that all the patches in the vicinity of the final result must be classified positively by the object detector. The N constraint requires that all other patches must be classified negatively by the object detector.

This section is organised as follows. In Section 4.3.1 we discuss how the results of tracker and the detector are combined into one, and then the criteria for output being valid is discussed. In Section 4.3.2 the PN constraints are discussed.

4.3.1 Fusion and Validity

The algorithm for fusion is given below:

1. If the detector gives exactly one result with confidence higher than the result from the tracker, then the result of the detector is selected. This equates to re-initialisation of the recursive tracker.
2. If the recursive tracker estimates a bounding box and is not re-initialised because of the point 1. Then the result of the recursive tracker is selected.
3. All other cases, no bounding box is selected, which suggests that object is not visible in the current scene.

The conditions under which selected final result is considered valid, both of the conditions assume that the tracker was not re-initialised by the detector, in all other cases the result is considered invalid are:

1. The final result is also valid if the previous result was valid and the recursive tracker produced a result with confidence larger than θ^+.
2. The final result is also valid if the previous result was valid and confidence of the recursive tracker output is greater than threshold θ^-.

The first bounding box is always valid. The threshold θ^+ indicates that a result belongs to the positive class and the threshold θ^- indicates that a result belongs to the negative class.

4.3.2 P/N - Learning

The learning method used in this algorithm is **semi-supervised learning**. In this type of learning there are both labelled examples as well as unlabelled data. The semi-supervised learning method here used the information present in the training data as supervisory information [28], the unlabelled data is distributed in class and the classifier is updated using the class separation as a training set. In our case there is exactly one labelled positive example which is first bounding box selected in the initial frame of the sequence.

Also here [3] the positive (P) and negative (N) constraints direct the learning restricting the labelling of unlabelled set. P-N learning evaluates classifier on the unlabelled data, removes the examples are that been classified in contradiction with the structural constraints and augments the training set with the corrected samples in a iterative process. *P-constraints* are used to identify examples that have been labelled negative by the constraint but the constraints require a positive label. *N-constraints* are used to identify examples that have been classified as positive but the constraints require negative label.

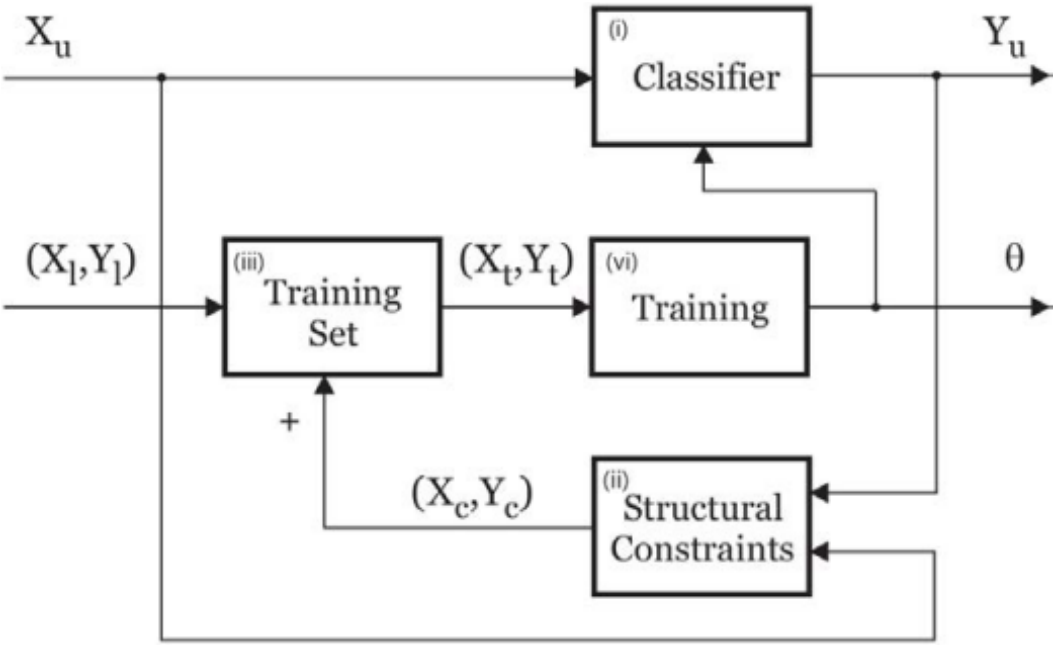

Figure 4.10. P/N Constraints. Image is from [16] .

Let x_i correspond to the training examples and the y_i to the corresponding class label. And, the classifier is a function $f : X \to Y$ that is applied to unseen data. In Figure 4.10., X_u refers to the unlabelled data available. This data is first classified by an existing classifier that assigns labels Y_u to X_u. Then, the structural constraints, according to some criterion, identify misclassified examples X_c with new labels Y_c. These examples are then added to the training set and classifier is updated.

The constraints used here are proposed in [3]. According to P-Constraints all the patches that are highly overlapping (default: $> 60\%$) with the final result must be classified as positive examples. and according to N-Constraint, all patches that are not overlapping (default: $< 20\%$) with the valid final result must be classified as negative examples. For measuring overlap, we employ the metric described in 6.1.1.

Chapter 5

Software

5.1 Architecture

The OpenTLD code is restructured from 4.2 to stream like architecture as shown in Figure 5.1. The information between the components are passed by storing information as meta-data in meta-data container.

During the first frame all the components are initialised. The Median flow tracker saves the location of the object in the first frame. The Patch variance filter saves the variance of selected patch, and uses halve the value as default for threshold to filter out patches from the second frame onwards. Ensemble Classifier and NN Classifier learn the first positive patch and few surrounding negative patches. In the first frame Non-maximal suppression, Clustering and hypothesis Fusion are inactive and the first selected box is the output.

During the next frames, the median flow tracker estimates the location of the object, based on the first selected bounding box. The Patch variance filter, Ensemble Classifier, NN Classifier and Non-Maximal suppression and clustering work as a single detector (i.e detector cascade). The detector detects the object based on the learning of object so far. The predicted output from tracker and detected object from detector is combined to single result in Hypothesis fusion. This final result is then validated. If the final result is valid, this output is saved in meta data container and learnt by the Ensemble Classifier and NN Classifier during the run in the next frame.

Stream like architecture facilitates:

- Evaluation of individual components, for example we could just use the Median flow tracking to track the object, analyse frame rate of individual components.
- Replacement of individual components of OpenTLD, for example we could

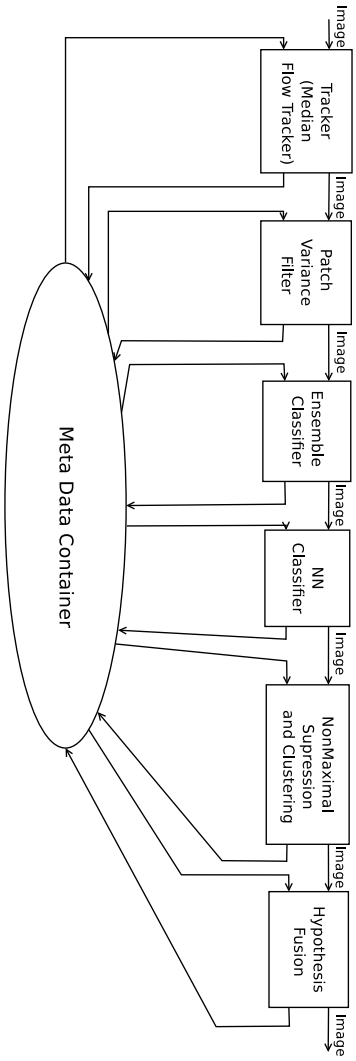

Figure 5.1. OpenTLD stream like architecture.

replace the detector cascade with OpenCV Eye detector for Eye tracking.

However, because of this new program work-flow, extra memory is needed to save the data from previous run.

5.1.1 Multi-Object Tracking

The OpenTLD algorithm is extended to track multiple objects. Each of components of the algorithm is duplicated (i.e. instantiated) as many number of times as the number of object to be tracked. The selected objects are either taken from the text file or received as meta-data. This is not very optimised way to track multiple objects, the common computations and memory for multiple object can be combined so that computations can be reduced and memory usage also could be reduced. During the evaluation we track maximum of two objects (i.e. two eyes).

OpenTLD should not work very well for 2 similar objects in the same sequence, because the negative training data is taken from the surroundings and classifier would become weaker if there are similar objects in the sequence. However it does not affect tracking 2 eyes because frontal appearance of both the eyes are different from each other, essentially they are the reflections of each other.

5.2 Evaluation Framework

The evaluation framework is shown below in Figure 5.2. The evaluation scripts written in python fetch the image sequence one by one and use the library API's to run the algorithm on the images. The output of the algorithm is collected, and the performance of the algorithm run is calculated. The evaluation scripts also keep track of the frame rate of the whole algorithm and individual components. The parameters of the algorithm can be configured via the configuration file. Finally the evaluation script gives the report of the run which can be analysed.

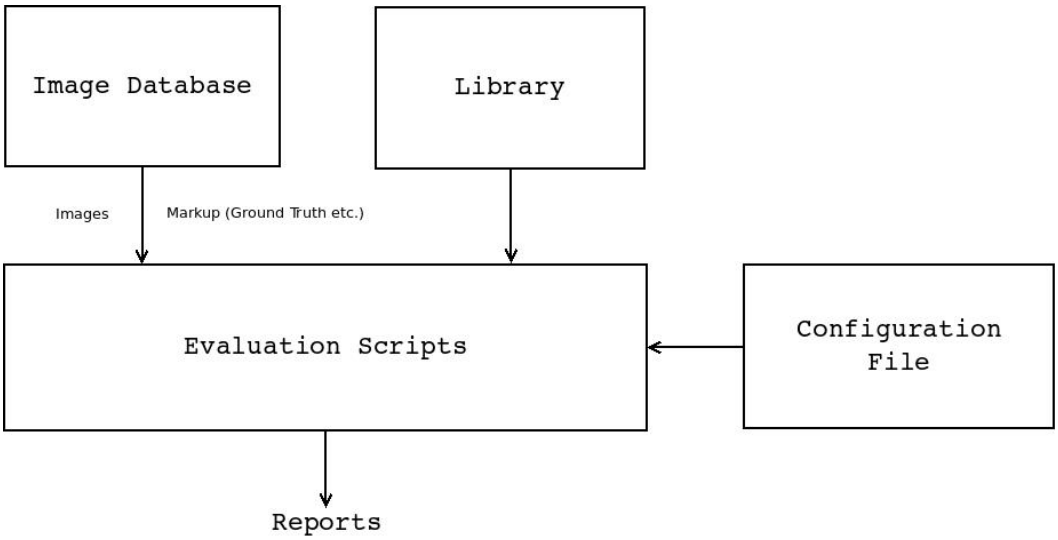

Figure 5.2. Evaluation Framework.

Chapter 6

Evaluation

6.1 Setup

All the experiments were run on Intel® Xeon® Processor E5430, which is a quad-core processor running at 2.66GHz. The operating system is Ubuntu 12.04 with kernel release 3.2.0-26-generic-pae.

The multi-threaded optimisation described in Section 4.2.1 is implemented using an OpenMP pragma in OpenTLD.

6.1.1 Measures to check the performance of object tracking algorithm

We use the measures as described in [16] to calculate the overlap, recall and precision. To measure the overlap between two boxes, we use the formula from PASCAL challenge [1]

$$overlap = \frac{B_1 \cap B_2}{B_1 \cup B_2} = \frac{I}{B_1 + B_2 - I}$$

where B_1 is the area of the first box, B_2 the area of second box and I is the area of the intersection of the two bounding boxes also as shown in Figure 6.2.

It is shown in [2] that this overlap measure equally penalizes translations in both directions and scale changes. Each frame of the sequence is categorised as one of the five possible cases shown in Figure 6.1 , based on the overlap between the algorithmic output and ground truth values.

A result is considered

- case a: **True Positive(TP)**: Overlap is larger than a threshold ω.

- case b: **False Negative(FN)**: The Ground truth exist and Algorithm yields no result.
- case c: **False Positive(FP)**: The Algorithm yields a result, but ground truth does not exist.
- case d: **False Negative and False Positive**: Overlap is lesser than the threshold ω
- case e: **True Negative(TN)**: Ground truth does not exist and Algorithm does not yield a result.

At the end of processing the sequence, all occurrences of TP, FP, TN and FN are counted. Based on these recall is defined as

$$recall = \frac{TP}{TP + FN} \tag{6.1}$$

recall measures the fraction of of positive examples that are correctly labelled [30]. Precision is defined as

$$precision = \frac{TP}{TP + FP} \tag{6.2}$$

For good performance, both precision and recall are important. Therefore, it is common to express the performance of object tracking algorithm by both precision and recall.

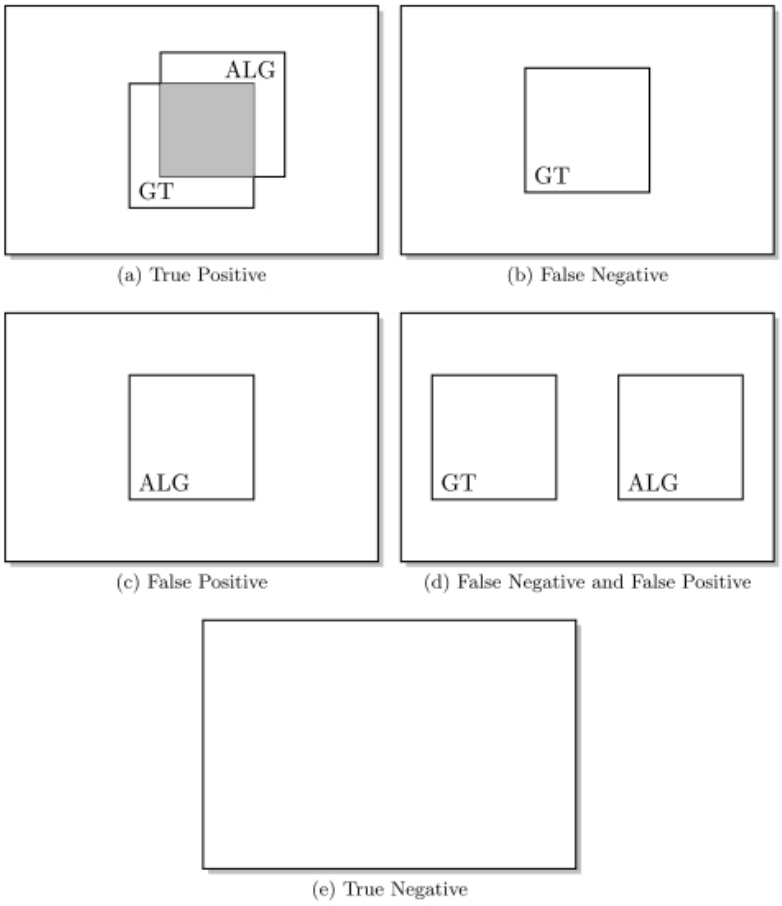

Figure 6.1. Five possible cases when comparing algorithmic results to the ground truth values. Image is from [16] .

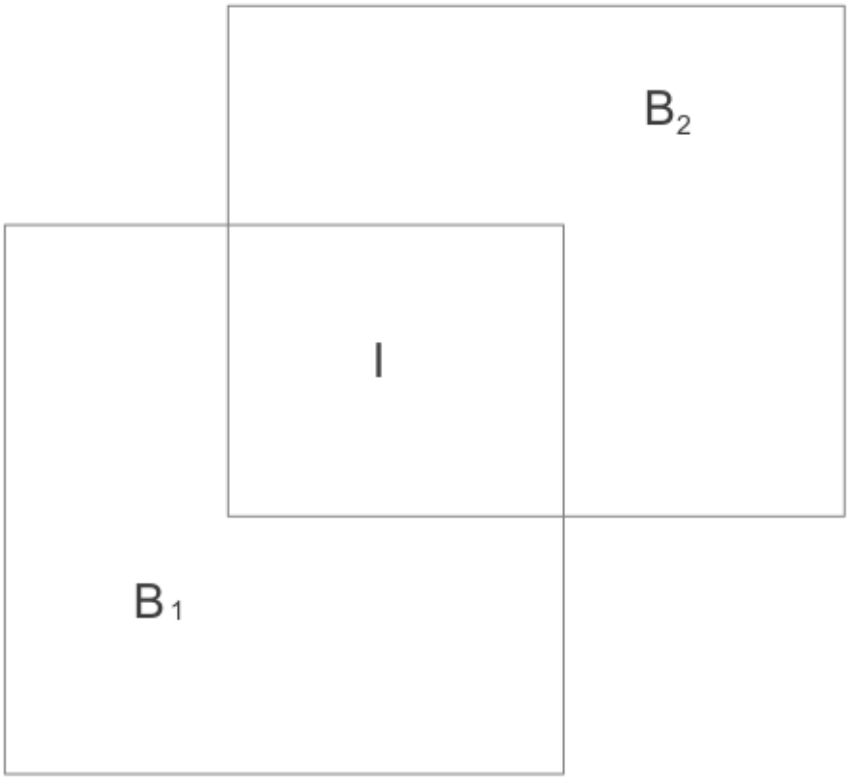

Figure 6.2. The overlap between two boxes B_1 and B_2 is the area of their Intersection I divided by area of the union $B_1 + B_2 - I$. Image is from [16] .

6.1.2 Memory Components

Memory that stays constant during runtime

As discussion in Section 4.2.1 sliding window based approach is used for object detection, sub-images of and input image are tested whether they contain the object of interest. The number of such sub images is restricted to

$$numWindows = \sum_{s \in 1.2^{-minScale...maxScale}} \lfloor (\frac{n - s(w + d_x)}{sd_x}) \rfloor \lfloor (\frac{m - s(h + d_y)}{sd_y}) \rfloor \tag{6.3}$$

Where $s = 1.2^a, a \in minScale...maxScale$, n = image width, m = image height, w width of initial bounding box, h = height of initial bounding box, d_x and d_y are

margins between two adjacent subwindows and set to be $\frac{1}{10}$ of the values of original bounding box.

For a image size of 320×240 and bounding box size 90×39, the numWindows = 11116.

Below are the significant memory components (variables in C++ code) of the algorithm.

1. **Tracker**:
   - Saved previous Image: Size $\propto$ Image Width $\times$ Image Height.
2. **Detector Cascade**:
   - Scales : Stores the scaling factors. Size $\propto$ Max Scale - Min Scale + 1. Default: Min Scale = -10 and Max Scale = 10.
   - Windows : Stores the scales of subwindows. Size $\propto$ TLD_WINDOW_SIZE $\times$ numWindows, Where by default TLD_WINDOW_SIZE = 5.
   - Window Offsets: Contains the offset for the subwindows. Size $\propto$ TLD_WINDOW_OFFSET_SIZE $\times$ numWindows. Where by default TLD_WINDOW_OFFSET_SIZE = 6.
   - Variance: Has the Variances of all subwindows for current frame. Size $\propto$ numWindows.
   - Posteriors: Posterior probability of each subwindow. Size $\propto$ numWindows.
   - Feature Vectors: Has the fern feature value of subwindows for the current frame. Size $\propto$ numWindows $\times$ numTrees. Default numTrees = 13.
3. **Patch Variance Filter**:
   - Integral image: Used for calculating the variance. Size $\propto$ Image Width $\times$ Image Height.
   - Squared integral image: Used for calculating the variance. Size $\propto$ Image Width $\times$ Image Height.
4. **Ensembler Classifier**:
   - FeatureOffsets: Has the offsets to the features points of each subwindow. Size $\propto$ numScales $\times$ numTrees $\times$ numFeatures. Default numScales: 0 $<$ numScales $<$ (Max Scale - Min Scale + 1). Default numTrees = 13, numFeatures = 10.
   - Features: contains the feature points of a single subwindow. Size $\propto$ 2 $\times$ 2 $\times$ numFeatures $\times$ numTrees, Default numTrees = 13, numFeatures = 10.

- Posteriors: Distribution of posterior probabilities Size $\propto$ numTrees $\times$ numIndices. Default numTrees = 13, numIndices = $2^{numFeatures}$.
- Positives: Keeps a count of number of times a pixel comparison was voted as positive. Size $\propto$ numTrees $\times$ numIndices. Default numTrees = 13, numIndices = $2^{numFeatures}$.
- Negatives: Keeps a count of number of times a pixel comparison was voted as negative. Size $\propto$ numTrees $\times$ numIndices. Default numTrees = 13, numIndices = $2^{numFeatures}$.

5. **Memory Introduced because of Stream like Architecture.**
   - Detector Cascade filter Index: Keeps detail whether a subwindow got filtered or not by the previous detector in detector cascade. Size $\propto$ numWindows .
   - Copy of Variance in NN Classifier.
   - Copy of Posterior in NN Classifier.
   - Saved previous Image in NN Classifer. Size $\propto$ Image Width $\times$ Image Height.

Memory that grows during run time

NN Classifer:

- Positively and Negatively Classified Patches: The NN Classifer keeps the templates of positively and negatively Classified Patches. This memory grows, and causes the NN Classifier to become slow as the number of positive and negative templates grow.

6.1.3 Description of sequences used for Evaluation

The sequence David Indoor consists of 761 frames and shows a person walking from an initially dark setting into a bright room. No occlusions occur in this sequence. Pedestrian 2 (338 frames) show pedestrians being filmed by an unstable camera. The sequence Car consists of 945 frames showing a moving car. This sequence is challenging because the images exhibit low contrast and various occlusions occur. They are part of TLD Dataset [16].

Eye_track_seq1, Eye_track_seq2, Eye_track_seq3, Eye_track_seq4, are part of eye tracking database and contains close view of person's face.

My_Eye_Seq1, My_Eye_Seq2 was created and ground truth was marked (i.e. eyes) to better evaluate OpenTLD algorithm. This contains a person's face close to

the camera, face moves away(i.e scale change), occlusion and face leaving the screen and re-entry.

Details of the sequences are given in the Table 6.1.

Sequence Name	Image size	Frames	Moving camera	Partial occlusions	Full occlusions	Out-of-plane rotation	Illumination change	Scale change	Similar objects
David Indoor	320x240	761	yes	yes	no	yes	yes	yes	no
Pedestrian 3	320x240	184	yes	yes	yes	no	no	no	yes
car	320x240	945	yes	yes	yes	no	no	no	yes
Eye_track_seq1	640x480	792	no	no	no	no	no	no	no
Eye_track_seq2	640x480	895	no	no	no	no	no	no	no
Eye_track_seq3	640x480	1419	no	no	no	no	no	no	no
Eye_track_seq4	640x480	1323	no	no	no	no	no	no	no
My_Eye_Seq1	640x480	1225	no	yes	yes	no	yes	yes	no
My_Eye_Seq2	640x480	1782	no	yes	yes	no	yes	yes	no

Table 6.1. Description of Sequences used for evaluation

In the evaluation left eye and the right eyes are tracked separately, (L) is used to refer to the left eye, (R) is used to refer to the right eye.

6.1.4 Assumption

- In some of the TLD sequences downloaded the ground truth is not good i.e. The ground truth box does not completely overlap with the object to be tracked. And, hence some of the true positives will be regarded as false positives and vice-versa. However, again such cases are very few and effect is acute.

6.1.5 Random Components of the Algorithm

There are two random components of the OpenTLD algorithm which causes the performance to vary, when the algorithm is run on the same sequence again and again.

1. Random points for Fern feature calculation in Ensemble Classifier, as discussed in Section 4.2.3.
2. Negative templates for NN Classifier are randomly chosen from the rest of the image during the first frame.

To evaluate the algorithm easily and check the performance, frame rate and memory for different parameter values, these two randomness components need to be fixed.

6.2 Experiments

6.2.1 Strict Initial Learning of negative templates in NN Classifier.

In this section we look at a way to fix the one of random components mentioned in Section 6.2.2. Instead of learning templates randomly from the image, we choose templates whose NCC (normalised correlation coefficient) value is closest to the initial bounding box.

Refer the Table 6.2. The worst and best performance for a sequence is obtained out of 6 runs. We see that performance of the algorithm is always better than the worst performance, except for My_Eye_Seq1(L), it is possible to get a worse performance in more number of runs. The performance is obtained with constant feature points as fixed in Section 6.2.2.

More work can be done to recognise the negative templates for the NN Classifier so that the performance of algorithm is at its best.

	Without Strict Learning		Strict Learning
	Worst Performance	Best Performance	Performance
Pedestrian 3	0.32/0.5	0.32/0.54	0.32/0.54
David Indoor	0.09/0.44	0.1/0.55	0.21/0.73
car	0.3/0.3	0.79/0.79	0.74/0.75
Eye_track_seq1(L)	0.95/0.95	0.95/0.95	0.95/0.95
Eye_track_seq1(R)	0.99/0.99	1/0.99	0.99/0.99
Eye_track_seq2(L)	0.93/0.99	0.99/0.99	0.99/0.99
Eye_track_seq2(R)	0.98/0.98	0.98/0.98	0.98/0.99
My_Eye_Seq1(L)	0.28/0.6	0.56/0.96	0.19/0.47
My_Eye_Seq1(R)	0.27/0.45	0.76/0.99	0.31/0.51
My_Eye_Seq2(L)	0.12/0.18	0.36/0.57	0.27/1
My_Eye_Seq2(R)	0.32/0.35	0.47/0.57	0.38/0.58

Table 6.2. With and Without Strict Initial Learning. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

	Without Constant Features		With Constant features
	Worst Performance	Best Performance	Performance
Pedestrian 3	0.32/0.46	0.95/1	0.32/0.54
David Indoor	0.09/0.41	0.84/0.91	0.21/0.73
car	0.53/0.53	0.86/0.86	0.74/0.75
Eye_track_seq1(L)	0.01/1	0.95/0.95	0.95/0.95
Eye_track_seq1(R)	0.97/0.99	1/0.99	0.99/0.99
Eye_track_seq2(L)	0.0007/1	0.0007/1	0.99/0.99
Eye_track_seq2(R)	0.98/0.99	1/0.98	0.98/0.99
My_Eye_Seq1(L)	0.28/0.6	0.75/0.95	0.19/0.47
My_Eye_Seq1(R)	0.15/0.95	0.67/0.96	0.31/0.51
My_Eye_Seq2(L)	0.11/0.17	0.31/0.99	0.27/1.0
My_Eye_Seq2(R)	0.3/0.37	0.58/0.81	0.38/0.58

Table 6.3. With and Without Constant Feature Points. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

6.2.2 Fixing the random points for fern feature calculation

The feature points are fixed by running many trials on the sequence Eye_track_seq2 and choosing the feature points, which gives a good performance for this sequence and many other eye sequences from eye tracking database.

See Table 6.3 for worst and best performance of the algorithm along with performance for the fixed constant feature points. The algorithm with this constant feature points perform better than the worst performance for most of the sequence.

	T8F11		T10F13		T14F17	
	worst	best	worst	best	worst	best
Pedestrian 3	0.32/0.43	0.94/0.98	0.32/0.35	0.32/0.35	0.32/0.51	0.94/1.0
David Indoor	0.22/0.76	0.78/0.82	0.03/0.23	0.56/0.76	0.42/0.57	0.86/0.91
car	0.33/0.34	0.73/0.74	0.3/0.3	0.78/0.77	0.69/0.69	0.86/0.86
Eye_track_seq1(L)	0.95/0.95	0.95/0.95	0.01/1	0.95/0.95	0.95/0.95	0.95/0.95
Eye_track_seq1(R)	1.0/0.99	1.0/0.99	0.97/0.99	1.0/0.99	1.0/0.99	1.0/0.99
Eye_track_seq2(L)	0.0007/1.0	0.94/0.93	0.0007/1.0	0.0007/1.0	0.0007/1.0	0.99/0.99
Eye_track_seq2(R)	1.0/0.98	1.0/0.98	1.0/0.98	1.0/0.98	1.0/0.98	1.0/0.98
My_Eye_Seq1(L)	0.38/0.63	0.72/0.96	0.21/0.45	0.49/0.68	0.378/0.99	0.75/1.0
My_Eye_Seq1(R)	0.17/0.26	0.69/0.92	0.18/0.31	0.67/0.98	0.16/0.23	0.74/0.98
My_Eye_Seq2(L)	0.14/0.2	0.25/0.85	0.12/0.18	0.43/0.63	0.08/0.82	0.27/1.0
My_Eye_Seq2(R)	0.23/0.26	0.59/0.7	0.15/0.4	0.58/0.82	0.25/0.35	0.46/0.57

Table 6.4. Performance analysis with different number of trees(T) and features(F). The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

The effect of randomness points used for fern feature calculation cannot be completely removed. The number of points is not sufficient to represent the entire patch for various objects.

However the effect of Randomness can be reduced by:

- Using a well uniformly distributed set of features to cover the entire patch, and if each of the classifier is guaranteed to be based on different set of features [4]. For eye tracking most of the times the object to tracked are similar, this could be used as an advantage.
- Increasing the number of feature and tree along with the above condition. See Section 6.2.3.

6.2.3 Effect of changing the number of features and trees.

In this section the number of trees and the number of features are varied and the performance, frame rate and memory are checked.

The performance comparison is given the Table 6.4, the worst and best performance out of 5 runs is displayed. The frame rate comparison is given in Table 6.5, the average frame rate out of 5 comparisons are given. The changing run time memory component, number of positive and negative templates in the NN Classifier is given in the Table 6.6, the value is average value over 5 runs.

Observations:

1. The performance is better with less number of trees and feature, because of

	T8F11	T10F13	T14F17
Pedestrian 3	22.514	21.91	21.22
David Indoor	16.28	17.88	13.574
car	20.74	20.93	20.06
Eye_track_seq1	2.80	3.02	2.17
Eye_track_seq2	3.94	4.26	2.95
My_Eye_Seq1	4.61	4.96	3.67
My_Eye_Seq2	4.87	5.15	3.96

Table 6.5. Average frame Rate analysis with different number of trees(T) and features(F). The value is given in frames per second.

	T8F11	T10F13	T14F17
Pedestrian 3	6.4/23.2	5.2/27.4	4.4/17.2
David Indoor	57.2/193.2	54/173.8	47.6/120.4
car	35.2/70.4	36.0/75.6	36.6/48.2
Eye_track_seq1(L)	5.2/18.6	4.2/18.8	4.4/16.6
Eye_track_seq1(R)	6/20.6	5.6/19.4	5.8/16.2
Eye_track_seq2(L)	5.4/8.4	1.0/5.0	11.8/9.2
Eye_track_seq2(R)	5/20	5/17.6	4.8/15.2
My_Eye_Seq1(L)	30.8/69.6	24.8/63.4	31.2/39.2
My_Eye_Seq1(R)	32.4/55.2	36.4/64.6	33.0/55.2
My_Eye_Seq2(L)	37.8/120.2	40.0/123.8	22.0/45.0
My_Eye_Seq2(R)	45.8/108.8	49.2/125.2	53.0/94.4

Table 6.6. NN Classifier memory analysis with different number of trees(T) and features(F). The first value in the cell is average number of positive templates and second value is average number of negative templates at the end of the run.

the shifted dependence to NN Classifier in Detector Cascade.

2. The performance is better with greater number of trees and features, because increased number of feature points in Ensemble Classifier causes the algorithm to fail less often.

3. The frame rate does not show much dependence on number of trees and features. However, average value shows decrease frame rate for T8F11 and T15F18 compared to default T10F13. Since better performance causes more number of positive and negative templates in NN Classifier, causing it to be slower.

4. The number of positive and negative templates shows direct relation with the performance. However, not true always, sometimes OpenTLD algorithm drifts and gives a lot of false positives.

6.2.4 Effect of changing image sizes

In this section the image size and the MinSize (the minimum size of the subwindow for window based scanning) are varied and the performance, frame rate and memory are checked.

The random components are fixed as mentioned in Section 6.2.1 and Section 6.2.2.

The performance comparison is given the Table 6.7. The frame rate comparison is given in Table 6.8. The changing run time memory component, number of positive and negative templates in the NN Classifier is given in the Table 6.9.

Image Size	640 x 480			320 x 240			160 x 120			120 x 90		
MinSize	25	15	5	25	15	5	25	15	5	25	15	5
Pedestrian 3	0.96/1	0.96/1	0.96/1	0.32/0.54	0.92/0.92	0.89/1.0	0.07/1	0.07/1	0.87/0.93	0.03/0.122	0.03/0.12	0.84/0.87
David Indoor	0.001/1	0.001/1	0.001/1	0.21/0.73	0.16/0.41	0.84/0.91	0.006/0.71	0.27/0.28	0.006/0.71	0.03/0.04	0.03/0.04	0.5/0.5
car	0.94/0.98	0.61/0.63	0.73/0.76	0.74/0.75	0.67/0.68	0.77/0.78	0.03/1	0.82/0.83	0.88/0.88	0.03/1	0.82/0.82	0.78/0.8
Eye_track_seq3(L)	0.71/1	0.59/1	0.59/1	0.74/0.78	0.77/0.79	0.89/1.0	0.71/0.71	0.71/0.71	0.71/0.71	0.29/0.29	0.24/0.24	0.67/0.67
Eye_track_seq3(R)	0.7/1	0.7/1	0.7/1	1.0/1.0	1.0/1.0	0.99/1	0.9/0.9	0.91/0.91	0.81/0.89	0.29/0.29	0.68/0.68	0.81/0.81
Eye_track_seq4(L)	0.93/0.93	0.4/0.4	0.4/0.4	0.17/0.17	0.77/0.77	0.17/0.17	0.001/0.001	0.001/0.001	0.001/0.001	0.001/1	0.001/1	0.001/1
Eye_track_seq4(R)	0.86/0.86	0.86/0.86	0.86/0.86	0.13/0.13	0.23/0.23	0.14/0.14	0.001/0.03	0.001/0.035	0.001/0.001	0.0007/1	0.001/1	0.005/1
My_Eye_Seq1(L)	0.19/0.44	0.36/1	0.38/0.99	0.55/0.99	0.35/0.72	0.37/1	0.008/1.0	0.32/0.36	0.26/0.3	0.008/1	0.17/0.2	0.2/0.25
My_Eye_Seq1(R)	0.31/0.51	0.4/0.75	0.43/0.77	0.31/0.36	0.35/0.4	0.5/0.58	0.08/0.98	0.51/0.57	0.29/0.33	0.0008/0.98	0.0008/1	0.17/0.22
My_Eye_Seq2(L)	0.27/1	0.25/0.96	0.24/0.95	0.14/0.72	0.25/0.84	0.15/0.22	0.0067/1.0	0.06/0.26	0.11/0.11	0.0005/1	0.0005/1	0.0005/1
My_Eye_Seq2(R)	0.38/0.48	0.26/0.37	0.33/0.45	0.33/0.37	0.28/0.32	0.21/0.24	0.02/0.95	0.19/0.21	0.2/0.21	0.006/1.0	0.006/1	0.41/0.38

Table 6.7. Performance analysis with different image sizes and MinSize of subwindow. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

Image Size	640 x 480			320 x 240			160 x 120			120 x 90		
MinSize	25	15	5	25	15	5	25	15	5	25	15	5
Pedestrian 3	9.28	6.98	4.09	23.59	16.15	8.56	38.26	36.15	19.35	33.34	28.18	20.31
David Indoor	8.08	3.58	3.59	19.43	9.74	4.92	29.65	18.41	10.93	26.89	21.35	11.77
car	11.22	8.71	7.88	21.16	17.36	10.89	33.62	25.06	18.55	39.71	25.73	21.67
Eye_track_seq3	2.9	1.73	1.54	5.93	3.85	2.45	7.63	6.84	4.52	8.7	7.6	5.45
Eye_track_seq4	3.67	2.28	1.19	7.54	5.57	2.82	9.86	9.03	5.57	12.81	10.83	7.42
My_Eye_Seq1	4.61	2.81	1.69	9.15	6.44	2.88	14.39	9.9	5.7	17.68	11.32	7.73
My_Eye_Seq2	5.02	2.79	1.58	10.01	6.59	2.92	16.16	9.81	5.56	18.43	14.21	9.04

Table 6.8. Frame rate analysis with different image sizes and MinSize of subwindow. The value is given in frames per second.

Image Size	640 x 480			320 x 240			160 x 120			120 x 90		
MinSize	25	15	5	25	15	5	25	15	5	25	15	5
Pedestrian 3	8/20	6/19	9/25	6/26	13/25	17/28	2/7	2/10	13/23	2/4	2/8	16/36
David Indoor	1/3	1/1	1/1	4/17	12/88	2/4	1/5	46/142	1/4	32/82	56/208	110/269
car	27/47	27/40	28/42	42/78	50/93	41/54	2/3	63/96	58/93	1/3	67/82	77/126
Eye_track_seq3(L)	6/9	5/7	5/7	6/8	7/10	4/8	6/4	8/9	9/11	5/3	7/8	9/11
Eye_track_seq3(R)	4/9	5/10	5/12	7/14	9/14	8/11	7/4	5/5	4/4	6/5	11/9	11/15
Eye_track_seq4(L)	10/21	11/25	11/26	8/5	11/11	12/33	1/4	1/6	2/5	1/1	1/2	1/6
Eye_track_seq4(R)	8/17	10/20	11/17	6/4	10/14	10/21	2/3	3/3	14/54	1/1	1/1	1/3
My_Eye_Seq1(L)	21/41	16/26	14/29	36/32	37/87	21/29	1/3	64/77	64/133	1/4	1/2	44/175
My_Eye_Seq1(R)	30/47	37/68	37/58	62/87	69/100	76/120	2/1	75/69	66/165	1/4	43/123	49/83
My_Eye_Seq2(L)	16/34	19/35	14/36	17/22	29/49	97/279	1/3	22/34	102/304	1/1	1/1	1/1
My_Eye_Seq2(R)	46/126	41/111	63/123	94/299	105/357	100/250	3/1	78/236	93/221	1/4	1/1	64/84

Table 6.9. NN Classifier memory analysis with different image sizes and MinSize of subwindow. The first value in the cell is number of positive templates and second value is number of negative templates at the end of the run.

xratio, yratio	0.5, 1.5	0.75, 2	1, 2.5
Eye_track_seq3(L)	0.36/1.0	0.71/1.0	0.97/1.0
Eye_track_seq3(R)	0.001/1.0	0.70/1.0	1.0/1.0
Eye_track_seq4(L)	0.42/0.42	0.93/0.93	0.44/0.44
Eye_track_seq4(R)	0.13/0.13	0.86/0.86	0.86/0.86

Table 6.10. Performance analysis with different bounding box size. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

Observations:

1. The performance is in general better with bigger image size, however sometimes performance becomes better (along with decreasing MinSize) with smaller image size.
2. For some of the sequences the performance remains quite good until the image size 120x90.
3. The frame rate increase as the image size is decreased.
4. The number of positive and negative templates show direct relation with the performance. However, not true always, when OpenTLD algorithm drifts and gives a lot of false positives.

6.2.5 Varying Bounding Box Sizes.

In this section we vary the initial bounding box size, and scale the ground truth accordingly.

For eye sequences from eye tracking database we derive the bounding box from the eyelid points. We get the xmin (x minimum), by choosing the minimum value of x coordinates and xmax as the max of x coordinates. Similarly we get ymin and ymax. The starting point of bounding box are (xmin,ymin). Width of the bounding box is (xmax - xmin) and height is (ymax - ymin).

We do not derive the ground truth from the iris because the ground truth would be different when iris is in the left corner of the eye and the iris is in the right corner, even though the position of eye is same.

The random components are fixed as mentioned in Section 6.2.1 and Section 6.2.2.

The performance comparison is given the Table 6.10 and 6.11. The frame rate comparison is given in Table 6.12 and 6.13. The changing run time memory component, number of positive and negative templates in the NN Classifier is given in the table 6.14 and 6.15.

Observations:

xratio, yratio	0.75,0.75	1,1	1.5,1.5	2,2
Pedestrian 3	0.22/0.67	0.32/0.54	0.95/0.94	0.73/0.83
David Indoor	0.002/0.22	0.21/0.73	0.54/0.63	0.02/1
Car	0.52/0.52	0.74/0.75	0.55/0.91	1/1
My_Eye_Seq1(L)	0.36/0.98	0.19/0.44	0.91/0.98	0.92/1.0
My_Eye_Seq1(R)	0.35/0.52	0.31/0.51	0.87/0.97	0.93/1.0
My_Eye_Seq2(L)	0.21/0.68	0.27/1.0	0.18/0.94	0.85/0.97
My_Eye_Seq2(R)	0.32/0.73	0.38/0.48	0.006/1.0	0.44/0.46

Table 6.11. Performance analysis with different bounding box sizes. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

xratio, yratio	0.5, 1.5	0.75, 2	1, 2.5
Eye_track_seq3	3.5	3.15	2.79
Eye_track_seq4	3.64	3.84	3.64

Table 6.12. Frame rate analysis with different bounding box sizes. The value is given in frames per second.

xratio, yratio	0.75,0.75	1,1	1.5,1.5	2,2
Pedestrian 3	24.79	19.88	21.27	20.63
David Indoor	18.11	18.04	14.51	24.31
Car	20.72	19.25	21.96	15.07
My_Eye_Seq1	4.45	4.78	3.89	4.45
My_Eye_Seq2	4.91	5.02	4.96	4.7

Table 6.13. Frame rate analysis with different bounding box sizes. The value is given in frames per second.

xratio, yratio	0.5, 1.5	0.75, 2	1, 2.5
Eye_track_seq3(L)	7/8	6/9	4/8
Eye_track_seq3(R)	1/3	4/9	6/8
Eye_track_seq4(L)	13/21	10/21	9/18
Eye_track_seq4(R)	13/27	8/17	7/10

Table 6.14. NN Classifier memory analysis with different bounding box sizes. The first value in the cell is number of positive templates and second value is number of negative templates at the end of the run.

xratio, yratio	0.75,0.75	1,1	1.5,1.5	2,2
Pedestrian 3	4/7	6/26	6/31	6/22
David Indoor	8/9	4/17	66/238	1/4
Car	31/75	42/78	45/87	83/267
My_Eye_Seq1(L)	13/30	21/41	45/60	27/42
My_Eye_Seq1(R)	44/48	30/47	53/112	25/49
My_Eye_Seq2(L)	20/49	16/34	11/31	38/85
My_Eye_Seq2(R)	26/51	46/126	1/1	59/122

Table 6.15. NN Classifier memory analysis with different bounding box sizes. The first value in the cell is number of positive templates and second value is number of negative templates at the end of the run.

1. The performance is in general better with bigger bounding box, but not always. If the object size is too small, object can't be represented well. If the object size is bigger the algorithm drifts.
2. The frame rate does not show dependence on bounding box size.
3. The number of positive and negative templates shows direct relation with the performance. Better the performance, the more the number of positive and negative templates in the NN Classifier.

6.2.6 Effect of reloading the positive patches.

In this section we keep reloading the positive templates in the NN Classifier for 5 runs and observe the behaviour.

The random components are fixed as mentioned in Section 6.2.1 and Section 6.2.2.

The performance comparison is given the Table 6.16. The frame rate comparison is given in Table 6.17. The changing run time memory component, number of positive and negative templates in the NN Classifier is given in the table 6.18.

Observations:

1. The performance for many sequence changes and remains constant. For few sequences it keeps varying.
2. The frame rate does not show dependence on run number.
3. The number of positive and negative templates increases because of reloading of the templates after each run.

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Pedestrian 3	0.32/0.54	0.32/0.5	0.32/0.5	0.32/0.5	0.32/0.5
David Indoor	0.21/0.73	0.21/0.73	0.21/0.73	0.21/0.73	0.21/0.73
car	0.74/0.75	0.41/0.41	0.68/0.69	0.68/0.69	0.68/0.69
Eye_track_seq3(L)	0.71/1.0	0.72/1.0	0.75/1.0	0.75/1.0	0.75/1.0
Eye_track_seq3(R)	0.7/1.0	0.7/1.0	0.7/1.0	0.7/1.0	0.7/1.0
Eye_track_seq4(L)	0.93/0.93	0.93/0.93	0.93/0.93	0.93/0.93	0.93/0.93
Eye_track_seq4(R)	0.86/0.86	0.86/0.86	0.86/0.86	0.86/0.86	0.86/0.86
My_Eye_Seq1(L)	0.19/0.44	0.21/0.31	0.14/0.17	0.17/0.21	0.18/0.22
My_Eye_Seq1(R)	0.31/0.51	0.16/0.23	0.17/0.24	0.24/0.33	0.25/0.31
My_Eye_Seq2(L)	0.27/1.0	0.15/0.23	0.12/0.18	0.14/0.21	0.15/0.21
My_Eye_Seq2(R)	0.38/0.48	0.29/0.31	0.29/0.31	0.3/0.31	0.3/0.31

Table 6.16. Performance analysis in different runs of the algorithm. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Pedestrian 3	30.93	34.04	34.03	33.83	34.24
David Indoor	26.55	26.85	26.19	26.41	26.11
car	30.13	30.35	27.85	29.62	30.27
Eye_track_seq3	6.04	5.9	5.88	5.79	5.91
Eye_track_seq4	7.04	7.16	7.02	6.96	7.07
My_Eye_Seq1	6.28	5.99	5.85	5.86	5.97
My_Eye_Seq2	6.12	5.48	5.41	5.39	5.58

Table 6.17. Frame rate analysis in different runs of the algorithm. The value is given in frames per second.

	1st Run	2nd Run	3rd Run	4th Run	5th Run
Pedestrian 3	6/26	8/34	8/36	8/36	8/36
David Indoor	4/17	4/15	4/15	4/15	4/15
car	42/78	82/111	102/114	102/115	102/115
Eye_track_seq3(L)	6/9	7/10	7/8	7/8	7/8
Eye_track_seq3(R)	4/9	5/9	5/9	5/9	5/9
Eye_track_seq4(L)	10/21	10/22	10/22	10/22	10/22
Eye_track_seq4(R)	8/17	9/20	9/21	12/20	12/21
My_Eye_Seq1(L)	21/41	49/93	82/149	110/164	141/221
My_Eye_Seq1(R)	29/47	62/84	76/83	93/102	119/92
My_Eye_Seq2(L)	16/34	62/184	95/190	113/218	146/214
My_Eye_Seq2(R)	46/126	133/271	172/318	222/349	233/333

Table 6.18. NN Classifier memory analysis in different runs of the algorithm. The first value in the cell is number of positive templates and second value is number of negative templates at the end of the run.

	Without Probable Patches	With Probable Patches
Pedestrian 3	0.32/0.54	0.33/0.5
David Indoor	0.21/0.73	0.21/0.73
car	0.74/0.75	0.75/0.77
Eye_track_seq3(L)	0.71/1.0	0.75/1.0
Eye_track_seq3(R)	0.70/1.0	0.76/1.0
Eye_track_seq4(L)	0.93/0.93	0.93/0.93
Eye_track_seq4(R)	0.86/0.86	0.86/0.86
My_Eye_Seq1(L)	0.19/0.47	0.56/0.80
My_Eye_Seq1(R)	0.31/0.51	0.74/0.95
My_Eye_Seq2(L)	0.27/1.0	0.27/0.98
My_Eye_Seq2(R)	0.38/0.48	0.39/0.48

Table 6.19. Performance analysis by loading the probable patches. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

6.2.7 Learning of probable patches

In this section we save the patches rejected by the NN Classifier for later consideration as shown in Figure 6.3. Once a new positive template is added to NN Classifier, the rejected patches are evaluated again as a candidate of positive template and added.

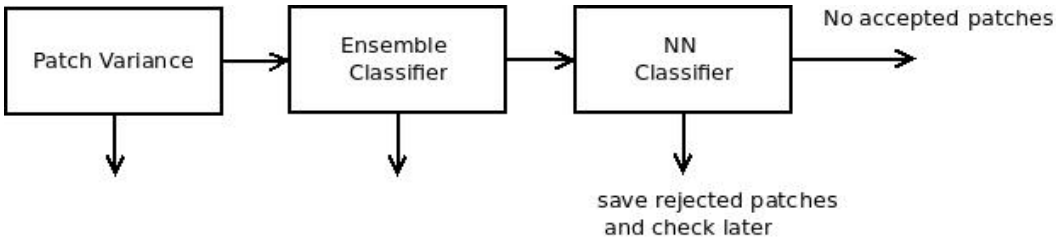

Figure 6.3. Loading the Probable Patches.

The random components are fixed as mentioned in Section 6.2.1 and Section 6.2.2.

The performance comparison is given the Table 6.19. The frame rate comparison is given in Table 6.20. The changing run time memory component, number of positive and negative templates in the NN Classifier is given in the Table 6.21.

Observations:

1. Learning of the probable patches increases the recall and precision.
2. The frame rate decreases.
3. The number of positive and negative templates increases because of loading

	Without Probable Patches	With Probable Patches
Pedestrian 3	19.88	21.44
David Indoor	18.04	16.21
car	19.25	17.88
Eye_track_seq3	3.15	2.88
Eye_track_seq4	3.84	3.7
My_Eye_Seq1	4.78	3.79
My_Eye_Seq2	5.02	3.89

Table 6.20. Frame rate analysis by loading the probable patches. The value is given in frames per second.

	Without Probable Patches	With Probable Patches
Pedestrian 3	6/26	14/31
David Indoor	4/17	4/17
car	42/78	73/75
Eye_track_seq3(L)	6/9	40/10
Eye_track_seq3(R)	4/9	20/9
Eye_track_seq4(L)	10/21	10/21
Eye_track_seq4(R)	8/17	8/17
My_Eye_Seq1(L)	21/41	150/121
My_Eye_Seq1(R)	30/47	155/112
My_Eye_Seq2(L)	16/34	104/89
My_Eye_Seq2(R)	46/126	118/152

Table 6.21. NN Classifier memory analysis by loading the probable patches. The first value in the cell is number of positive templates and second value is number of negative templates at the end of the run.

of probable patches.

6.2.8 Effect of loading the reflection of left on the right eye and vice versa.

In this section we load the reflection of left eye as a positive template to NN Classifier for the right eye and vice versa.

The random components are fixed as mentioned in Section 6.2.1 and Section 6.2.2.

The performance comparison is given the Table 6.22, in the table we also show the performance of detector cascade alone in detecting the object. The frame rate comparison is given in Table 6.23. The changing run time memory component, number of positive and negative templates in the NN Classifier is given in the Table 6.24.

	NoRD	RD	NoRTLD	RTLD
Eye_track_seq1(L)	0.81/0.91	0.92/0.94	0.95/0.95	0.95/0.95
Eye_track_seq1(R)	0.97/0.99	0.98/0.99	1.0/0.99	1.0/0.99
Eye_track_seq2(L)	0.91/0.99	0.91/0.99	0.99/0.99	0.99/0.99
Eye_track_seq2(R)	0.99/0.99	0.99/0.99	1.0/0.98	1.0/0.98
Eye_track_seq3(L)	0.55/1.0	0.52/1.0	0.71/1.0	0.70/1.0
Eye_track_seq3(R)	0.40/1.0	0.52/1.0	0.70/1.0	0.71/1.0
Eye_track_seq4(L)	0.0/0.0	0.0/0.0	0.93/0.93	0.93/0.93
Eye_track_seq4(R)	0.0/0.0	0.0/0.0	0.86/0.86	0.86/0.86
My_Eye_Seq1(L)	0.16/0.5	0.22/0.67	0.19/0.44	0.27/0.60
My_Eye_Seq1(R)	0.27/0.6	0.27/0.59	0.31/0.51	0.32/0.52
My_Eye_Seq2(L)	0.25/1.0	0.24/0.0.95	0.27/1.0	0.25/0.94
My_Eye_Seq2(R)	0.20/0.43	0.21/0.44	0.38/0.48	0.38/0.48

Table 6.22. Performance analysis by loading the reflection. NoRD (No Reflection Detector Cascade Performance), RD (With Reflection Detector Cascade Performance), NoRTLD (No Reflection OpenTLD Performance), RD (With Reflection OpenTLD Performance). The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

	NoRTLD	RTLD
Eye_track_seq1	3.24	3.19
Eye_track_seq2	4.32	4.23
Eye_track_seq3	3.59	3.54
Eye_track_seq4	3.25	3.24
My_Eye_Seq1	4.78	4.51
My_Eye_Seq2	5.02	4.83

Table 6.23. Frame rate analysis by loading the probable patches. NoRTLD (No Reflection OpenTLD Performance), RD (With Reflection OpenTLD Performance). The value is given in frames per second.

Observations:

1. In many of the cases, loading of the reflection increases the performance (both whole algorithm as well as the detector cascade). However, in few cases in could also deteriorate the performance because of wrong learning, anti-symmetry, not enough frames to test the complete positive templates.
2. The frame rate decreases.
3. The number of positive and negative templates increases because of loading of reflection.

Doing this work specially when the face is frontal or the image sequence is long and consists of many variations of left and right eye. There is not much point in loading

	NoRTLD	RTLD
Eye_track_seq1(L)	5/14	10/15
Eye_track_seq1(R)	6/14	10/18
Eye_track_seq2(L)	19/17	23/35
Eye_track_seq2(R)	5/11	13/11
Eye_track_seq3(L)	6/9	7/9
Eye_track_seq3(R)	4/9	9/13
Eye_track_seq4(L)	10/21	16/23
Eye_track_seq4(R)	8/17	16/18
My_Eye_Seq1(L)	21/41	52/75
My_Eye_Seq1(R)	30/47	54/52
My_Eye_Seq2(L)	16/34	45/81
My_Eye_Seq2(R)	46/126	64/129

Table 6.24. NN Classifier memory analysis by loading the reflection. NoRTLD (No Reflection OpenTLD Performance), RD (With Reflection OpenTLD Performance). The first value in the cell is number of positive templates and second value is number of negative templates at the end of the run.

the reflection, if we do not encounter that appearance for the other eye in the image sequence.

6.2.9 Long run and frame rate variations

The algorithm is run on a long sequence (TLD_long_seq1_640_480) with 32765 frames and memory of NN Classifier continued to increase without restraint as shown in Figure 6.4 and 6.5. However, the size of one on this template is quite small, by default its 15x15 pixels in size.

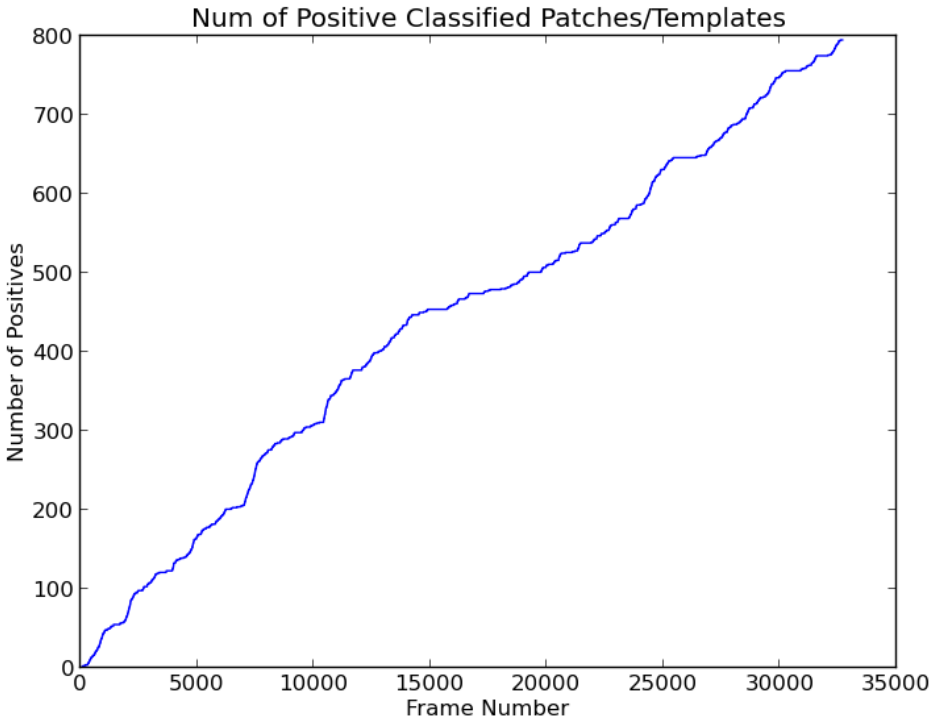

Figure 6.4. Positively Classified Patches.

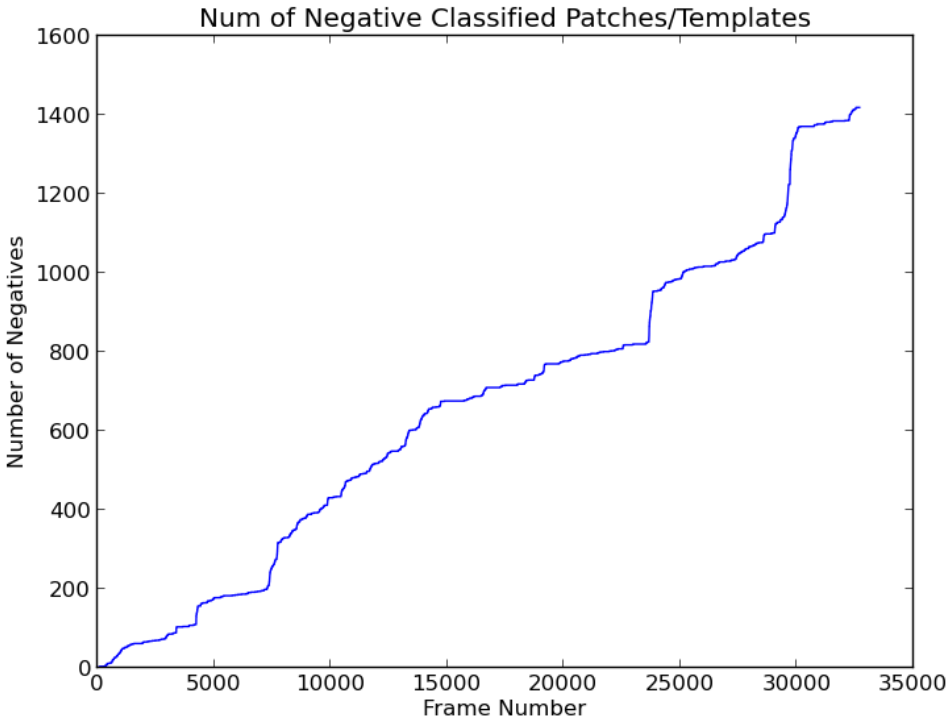

Figure 6.5. Negatively Classified Patches.

When the number of templates in NN Classifier exceeds some threshold, TLD [4] uses random forgetting of templates.

The frame rate variations are passed through a moving average filter of length 501.

Frame rate variation of OpenTLD algorithm is given in Figure 6.6. The whole OpenTLD algorithm starts to become slower as it runs for more time. This is because NN Classifier is becoming slower because of addition of large number of positive and negative templates.

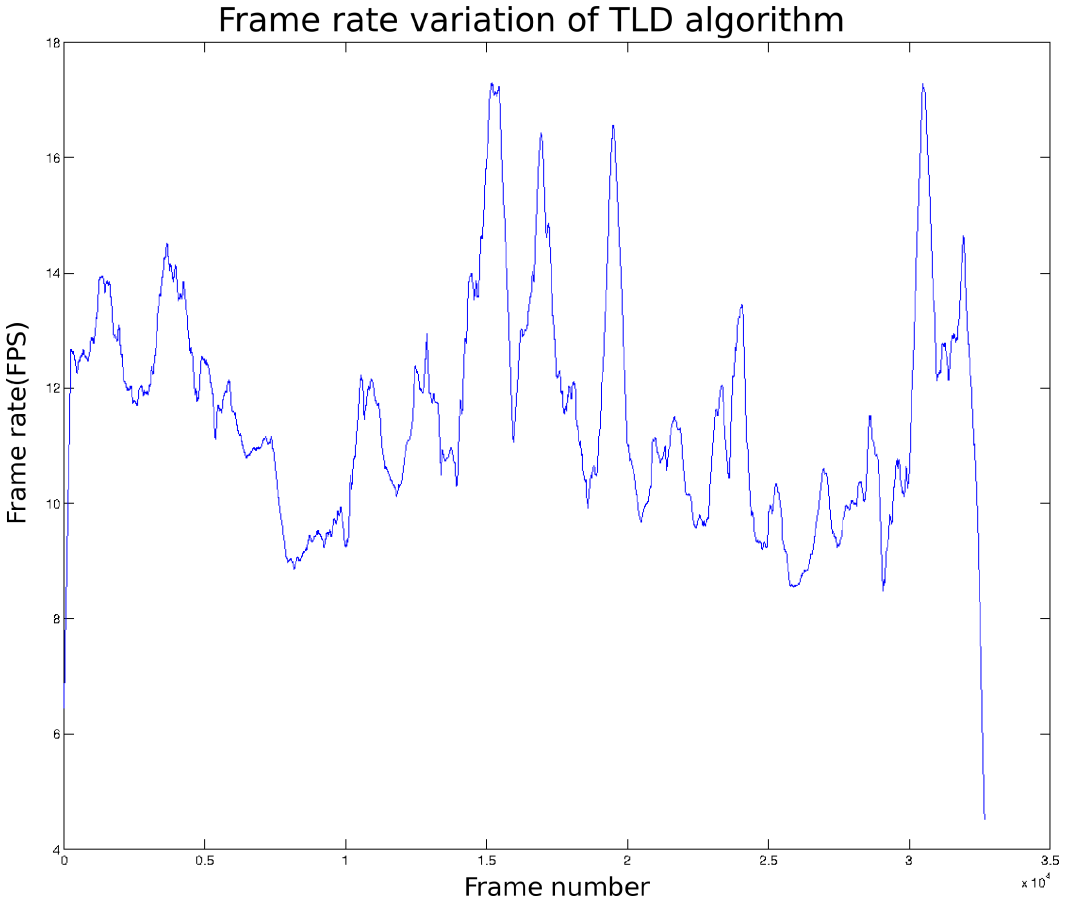

Figure 6.6. Variation of the frame rate of OpenTLD algorithm.

Frame rate variation of Patch Variance Filter is given in Figure 6.7, it does not decrease with time.

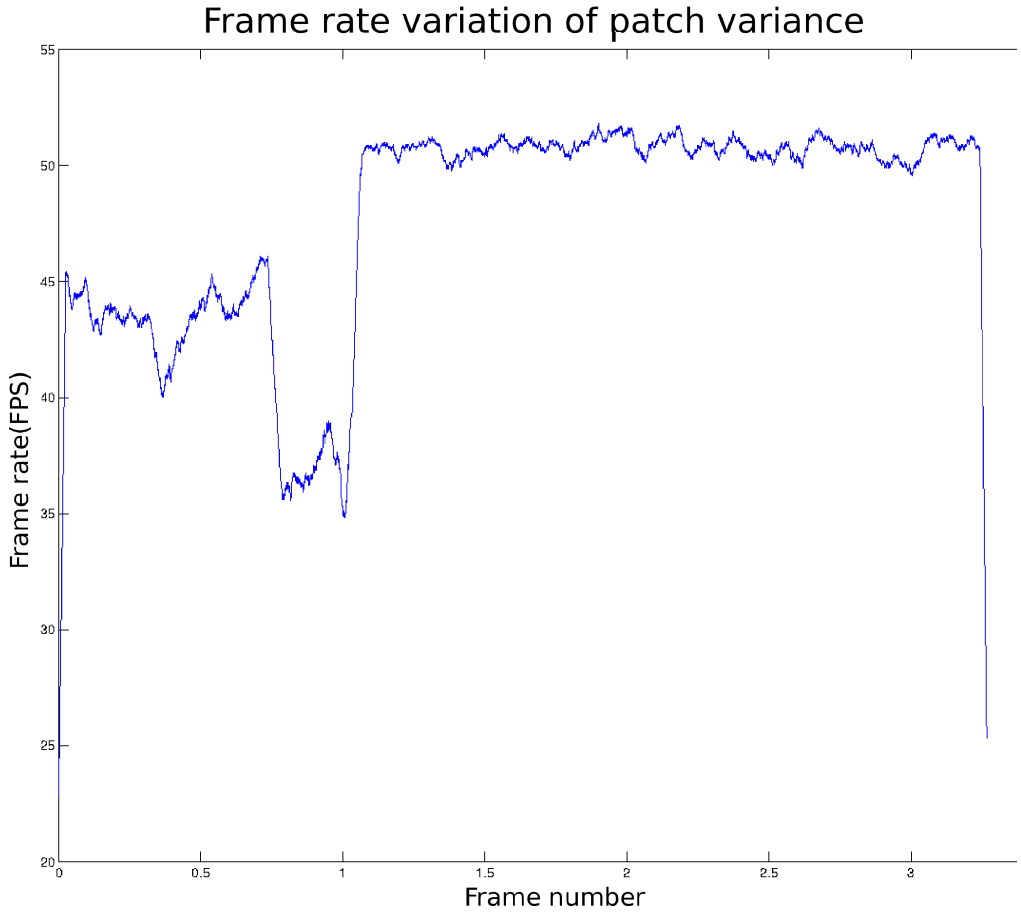

Figure 6.7. Variation of the frame rate of Patch Variance Filter.

Frame rate variation of Ensemble Classifier Filter is given in Figure 6.8, it does not decrease with time.

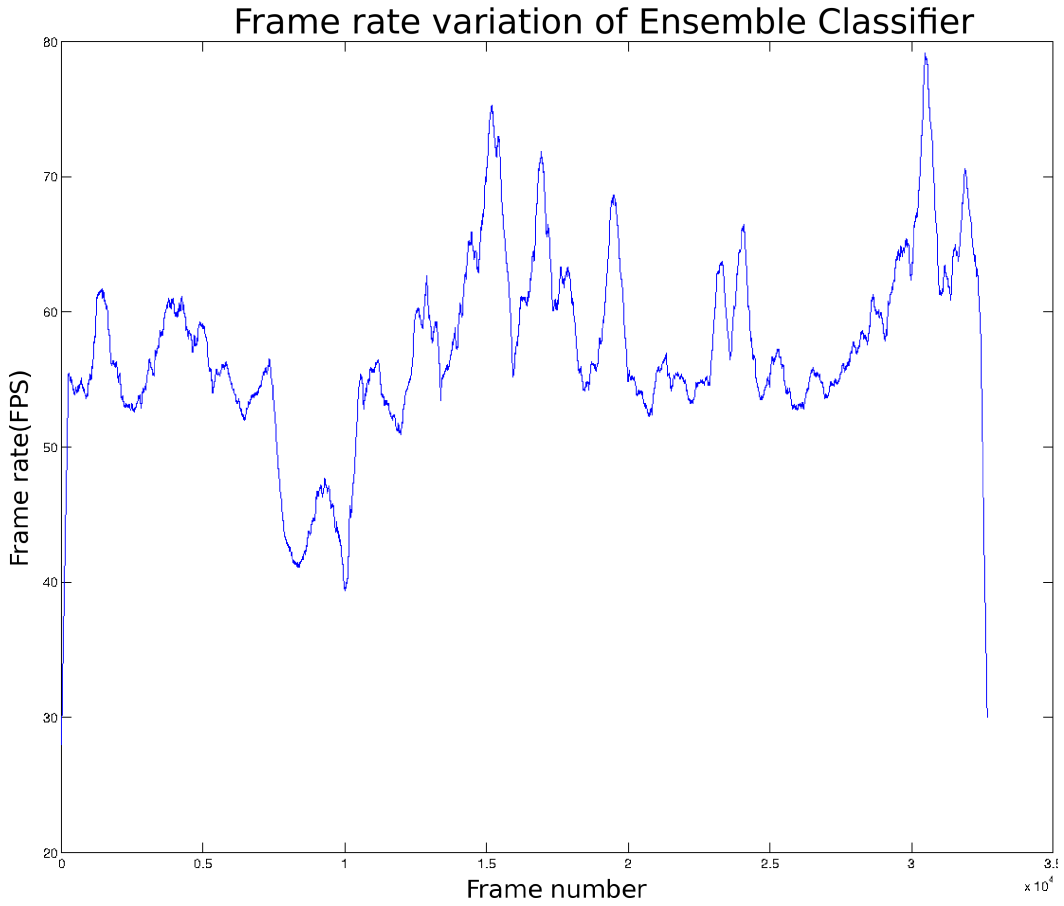

Figure 6.8. Variation of the frame rate of Ensemble Classifier.

Frame rate variation of NN Classifier is given in fig 6.9 NN Classifier is becoming slower because of addition of large number of Positive and Negative Templates.

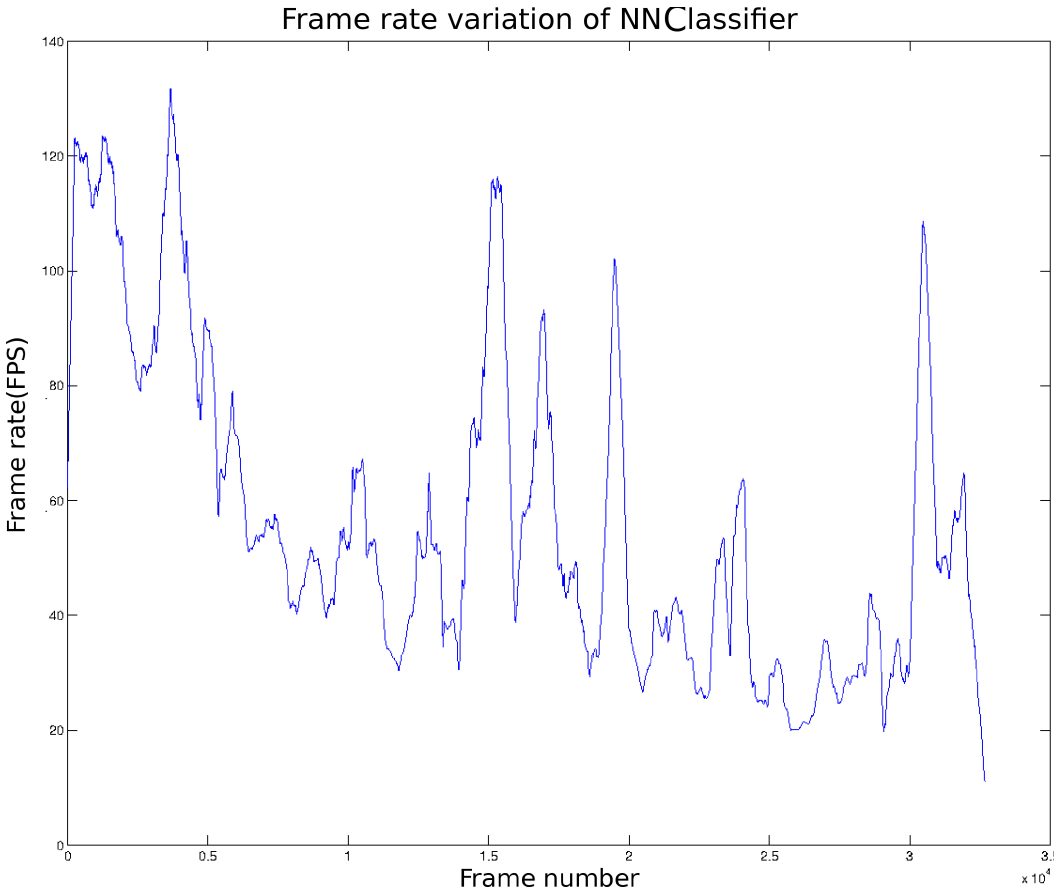

Figure 6.9. Variation of the frame rate of NN Classifier.

Frame rate variation of Non Maximal Suppression and Clustering is given in fig 6.10, this component is becoming slower because greater number of subwindows is being passed by the Detector Cascade.

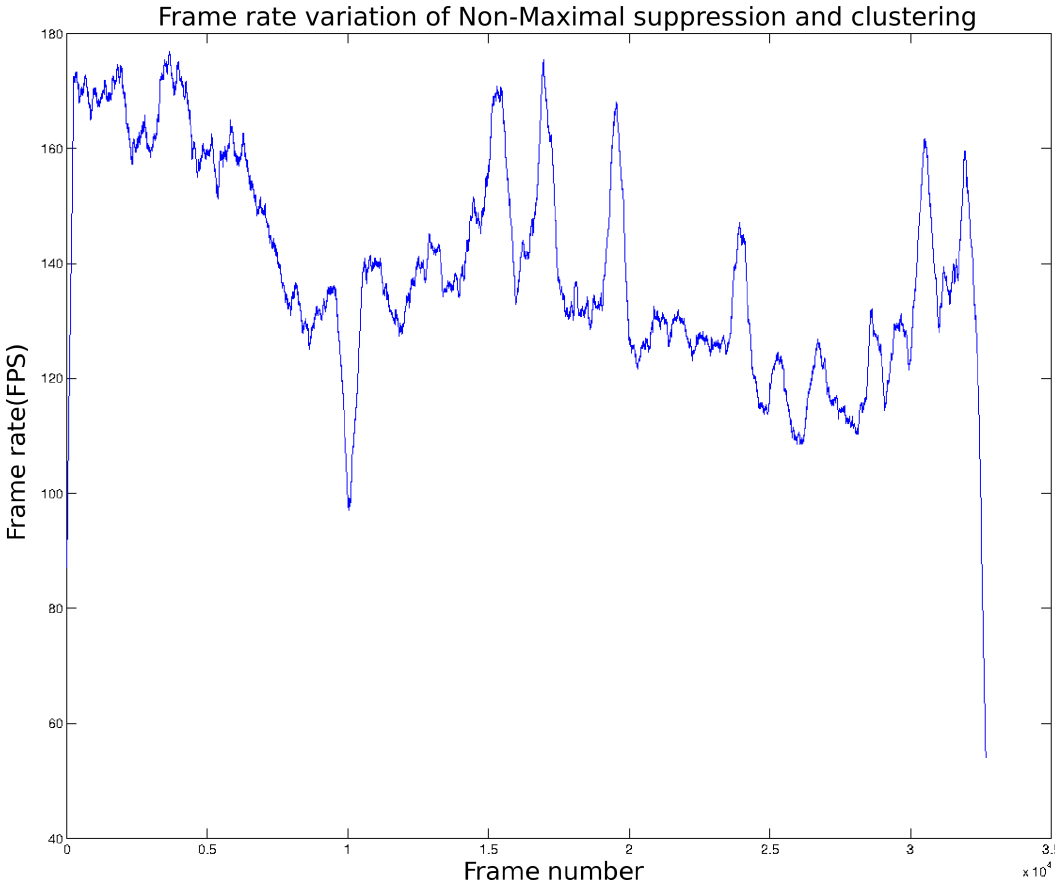

Figure 6.10. Variation of the frame rate of Non Maximal Suppression and Clustering.

	OpenTLD(Worst)	OpenTLD(best)	MFTracker	H_BP_MS	MS	EyeDetector
Pedestrian 3	0.32/0.35	0.94/1.0	0.31/0.26	0.02/0.02	0.94/0.79	-
David Indoor	0.03/0.23	0.86/0.91	0.88/0.88	0.06/0.06	0.009/0.009	-
car	0.3/0.3	0.86/0.86	0.03/0.03	0.008/0.007	0.09/0.08	-
Eye_track_seq1(L)	0.95/0.95	0.95/0.95	0.95/0.95	0.02/0.02	0.99/0.98	1.0/0.97
Eye_track_seq1(R)	0.97/0.99	1.0/0.99	1.0/0.99	0.01/0.01	1.0/0.99	1.0/0.99
Eye_track_seq2(L)	0.0007/1.0	0.99/0.99	1.0/0.99	0.007/0.007	1.0/0.98	0.46/0.88
Eye_track_seq2(R)	1.0/0.98	1.0/0.98	1.0/0.98	0.004/0.004	1.0/0.98	0.01/0.36
Eye_track_seq3(L)	0.1/1.0	0.9/1.0	1.0/1.0	0.01/0.01	0.76/0.76	1.0/1.0
Eye_track_seq3(R)	0.08/1.0	0.89/0.97	1.0/1.0	0.04/0.04	0.65/0.65	1.0/1.0
Eye_track_seq4(L)	0.4/0.4	0.93/0.93	0.4/0.4	0.003/0.003	0.003/0.003	0.0/0.0
Eye_track_seq4(R)	0.38/0.38	0.86/0.86	0.33/0.33	0.003/0.003	0.02/0.02	0.0/0.0
My_Eye_Seq1(L)	0.21/0.45	0.75/1.0	0.5/0.5	0.07/0.0	0.42/0.42	0.82/0.94
My_Eye_Seq1(R)	0.16/0.23	0.74/0.98	0.6/0.6	0.07/0.07	0.59/0.59	0.76/0.91
My_Eye_Seq2(L)	0.08/0.82	0.43/0.63	0.35/0.35	0.11/0.11	0.42/0.42	0.37/0.59
My_Eye_Seq2(R)	0.15/0.4	0.59/0.7	0.37/0.37	0.1/0.1	0.59/0.59	0.38/0.66

Table 6.25. Performance analysis of various algorithms. The first value in a cell denotes recall, the second value is precision. For $\omega = 0.5(50\%Overlap)$.

	OpenTLD	MFTracker	H_BP_MS	MS	EyeDetector
Pedestrian 3	21.78	135.14	244.29	251.81	-
David Indoor	15.91	127.65	236.0	200.28	-
car	20.57	138.26	234.55	167.8	-
Eye_track_seq1	2.66	24.37	21.87	44.54	13.22
Eye_track_seq2	3.71	27.01	25.11	44.54	23.79
Eye_track_seq3	4.27	23.36	21.46	28.46	11.85
Eye_track_seq4	8.46	25.08	23.97	45.4	25.18
My_Eye_Seq1	4.41	41.16	30.16	57.65	7.63
My_Eye_Seq2	4.66	39.9	32.16	51.54	15.02

Table 6.26. Frame rate analysis of various algorithms. The value is given in frames per second.

6.2.10 Comparison with other Trackers and Detector

In this Section we compare the performance of various Algorithms 1) OpenTLD 2) MFTracker (Median Flow Tracker) 3) H_BP_MS (OpenCV Mean Shift detector based on back projection of Histogram) 4) MS (Mean Shift Detector based on [29] 5) EyeDetector (Eye Detector from OpenCV)

The performance comparison is given the Table 6.25. The frame rate comparison is given in Table 6.26.

Observations:

1. MFTracker works well until the object disappears from the screen or is lost, no re-detection. And, the frame rate is high.

2. MS works also well, but fails for some sequences. The frame rate is higher.
3. H_BP_MS does not perform well.
4. EyeDetector performs well and gives good frame rate compared to OpenTLD but fails when the eyes are smaller. Eye Detector output is shaky.
5. In OpenTLD, Object can drift away in longer sequence and eyes won't be detected, when eyes are easily recognizable by OpenCV eye detector.

Looking at the good performance of EyeDetector at close range, we can possibly replace the EyeDetector instead of Detector Cascade for eye tracking.

And also median flow tracker should be used directly instead of the whole OpenTLD algorithm, when, object occlusion does not occur.

Chapter 7

Results and Conclusions

In this work we evaluated the OpenTLD algorithm, which is a novel approach to robust object tracking based on Tracking-Learning-Detection paradigm. We modified the algorithm for multi object tracking and modified it to the stream like architecture to facilitate the evaluation of individual component and to facilitate plug-in and plug-out of modules.

First, we saw the random components of the algorithm and saw how their influence can cause the algorithm to fail completely on one of the multiple runs of the algorithm. Then, we discussed the ways to reduce complete failure. Instead of learning the templates randomly from the image, we choose templates whose NCC value is closest to the initial bounding box. Then, we saw the ways to reduce the failure due to the randomness from the random points used in for fern feature calculation.

Then we saw what happens with changing image sizes, in general the algorithm performs very well with bigger image size, however, sometimes the performance of becomes better with a smaller image size. Also, the frame rate increases with decrease in the image size. We then saw that performance of the algorithm is better with bigger bounding box, but should be chosen based on usability, since too big bounding box can cause the tracking to drift (i.e. if bounding box covers the some of the background also).

The algorithm does not show any increase in the performance by re-running the algorithm by loading the positive samples, cause the object learning can be wrong and in that case object detected in the rerun can be not as good as the earlier run of the algorithm.

We then saw the robustness of the algorithm can be increased by learning the probable patches, with decrease in the frame rate and increase in the memory usage. Also, we saw that robustness of the algorithm can be increased by loading the reflection of left eye as a positive template to the right eye and vice versa can increase the performance of the algorithm.

We ran the algorithm on a very long sequence the positive and negative templates in the NN Classifier continue to increase without restraint. The TLD[4] algorithm forgets the templates randomly when the number of templates exceed a threshold. And then we looked at the way the frame rates of various components change during a long run of the algorithm.

Then the performance and frame rate of OpenTLD is compared with other algorithms. Median Flow Tracker alone works very well if there is no object occlusion. OpenCV EyeDetector performs well for the cases when the eyes are closer to the camera, else fails. This presents a possibility of replacing the detector cascade with OpenCV EyeDetector for eye tracking.

Finally the behaviour of the algorithm is checked by varying various parameters. The performance of the algorithm is quite sensitive to its parameters. Changing a little bit can cause the algorithm to perform very well or fail completely for a sequence.

Overall the OpenTLD algorithm is very good algorithm and many of the ideas in TLD paradigm are quite efficient, with few shortcoming such as the random components and drifting of the algorithm. The algorithm could be made more robust by correcting these shortcomings, which can be seen as possible future scope of the thesis.

7.1 Future scopes

Below presented is the possible future scope of work that can be done on the algorithm.

- Warping: In contrast to TLD [4], OpenTLD does not use image warping for learning. In TLD the initial detector is trained using labelled examples that generated as follows. The positive training examples are synthesized from the initial bounding box. First, 10 subwindows that are have maximal overlap with the initial selected patch are selected, and then for each of the bounding boxes, 20 warped versions are generated. Using warping for learning can make the performance of OpenTLD better.
- Using 3D depth data as input to Algorithm: Instead of passing RGB images to object tracking algorithm. We could pass the depth data (For example: depth data from Kinect Sensor) of a scene and check if the algorithm performs any better.
- Handling Out of Plane Rotation [4] In case of out-of-plane rotation, Median Flow tracker drifts from the target. The tracker typically stays away until the detector re-initializes its position to previously seen appearance. See [4] for more details.

- Current implementation of TLD trains only the detector and the tracker stay fixed. As a result the tracker makes always the same errors. An interesting extension would be to train also the tracking component [4].
- Multi-target tracking opens interesting questions how to jointly train the models and share features in order to scale [4]. Current implementation does not do this.
- 3D tracking on RGBD data (i.e. using Kinect sensors instead of camera): Instead of tracking the object in 2D world the TLD framework could be applied to tracking object in 3D world.
- Learning shape [31]: When we have collected many possible views of the object(i.e. positive templates in NN Classifier). These views could be combined to learn the shaped of the object.
- Parallelize the tracker and detector to increase speed: Currently, the detector runs after tracker is finished, however, the detector can run parallel to the tracker.
- Apply TLD framework to different combination of trackers and detectors [31]: the object tracking algorithm tracks any arbitrary object, however, based on the application domain different combination of tracker and detector could be more efficient.
- As there has been a huge research in computer vision focused on features, there is potential in describing the object appearance in a different feature space (e.g. Sift, HoG, Dot, LBP) which may lead to more robust tracking [32].
- A GPU implementation is possible for parallel analysis of subwindows during the detection [32].
- NN Classifier's Positive and Negative example space can be modified, can also optimise the object manifold. We noticed sometimes the algorithm performs better with just few numbers of examples. Hence, it should be possible to remove the positive and negative examples keeping the performance of the algorithm same.
- Automatic Initialization [31]: for the current algorithm the object needs to be specified in the first frame, however, when it's known which object are we tracking, we could just detect the object in the first frame and continue to track it.
- Eliminate planarity assumption [31].
- Design of more sophistical PN constraints for learning [3].
- Background subtraction [4] In case static scene, background subtraction can be used to improve the tracking capabilities.

Bibliography

[1] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2), 303-338.

[2] Matthews, L., Ishikawa, T., Baker, S. (2004). The template update problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions* on, 26(6), 810-815.

[3] Kalal, Z., Matas, J., Mikolajczyk, K. (2010, June). Pn learning: Bootstrapping binary classifiers by structural constraints. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (pp. 49-56). IEEE.

[4] Kalal, Z., Mikolajczyk, K., Matas, J. (2012). Tracking-learning-detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions* on, 34(7), 1409-1422.

[5] Yang, H., Shao, L., Zheng, F., Wang, L., Song, Z. (2011). Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18), 3823-3831.

[6] Wang, Q., Chen, F., Xu, W., Yang, M. H. (2011). An experimental comparison of online object-tracking algorithms. *SPIE: Image and Signal Processing*, 81381A-81.

[7] Yilmaz, A., Javed, O., Shah, M. (2006). Object tracking: A survey. *Acm Computing Surveys* (CSUR), 38(4), 13.

[8] Ballard, D. H., Brown, C. M. Computer vision. 1982.

[9] Black, M. J., Jepson, A. D. (1998). Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision*, 26(1), 63-84.

[10] Jepson, A. D., Fleet, D. J., El-Maraghi, T. F. (2003). Robust online appearance models for visual tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10), 1296-1311.

[11] Lim, J., Ross, D., Lin, R. S., Yang, M. H. (2004). Incremental learning for visual tracking. Advances in neural information processing systems, 17, 793-800.

[12] Avidan, S. (2007). Ensemble tracking. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 29(2), 261-271.

[13] Tobii Technologies. What is Eye Tracking?. Retrieved from *http://www.tobii.com*

[14] Wikimedia Foundation, Inc. Eye Tracking. Retrieved from *http://en.wikipedia.org*

[15] Wikimedia Foundation, Inc. Psycholinguistics. Retrieved from *http://en.wikipedia.org*

[16] Nebehay, G., Micusik, B., Picus, C., Pflugfelder, R. (2011). Evaluation of an online learning approach for robust object tracking.

[17] Lucas, B. D., Kanade, T. (1981, April). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence.*

[18] Shi, J., Tomasi, C. (1994, June). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference* on (pp. 593-600). IEEE.

[19] Kalal, Z., Mikolajczyk, K., Matas, J. (2010, August). Forward-backward error: Automatic detection of tracking failures. *In Pattern Recognition (ICPR), 2010 20th International Conference on* (pp. 2756-2759). IEEE.

[20] Viola, P., Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (Vol. 1, pp. I-511). IEEE.

[21] Dalal, N., Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (Vol. 1, pp. 886-893). IEEE.

[22] Schneiderman, H. (2004, June). Feature-centric evaluation for efficient cascaded object detection. In *Computer Vision and Pattern Recognition,*

2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on (Vol. 2, pp. II-29). IEEE.

[23] Ozuysal, M., Fua, P., Lepetit, V. (2007, June). Fast keypoint recognition in ten lines of code. *In Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on* (pp. 1-8). IEEE.

[24] Lampert, C. H., Blaschko, M. B., Hofmann, T. (2008, June). Beyond sliding windows: Object localization by efficient subwindow search. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (pp. 1-8). IEEE.

[25] Lepetit, V., Lagger, P., Fua, P. (2005, June). Randomized trees for real-time keypoint recognition. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (Vol. 2, pp. 775-781). IEEE.

[26] Blaschko, M. (2011). Branch and bound strategies for non-maximal suppression in object detection. In *Energy Minimization Methods in Computer Vision and Pattern Recognition* (pp. 385-398). Springer Berlin/Heidelberg.

[27] Murtagh, F. (1983). A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4), 354-359.

[28] Chapelle, O., Schölkopf, B., Zien, A. (2006). *Semi-supervised learning* (Vol. 2). Cambridge, MA:: MIT press.

[29] Comaniciu, D., Ramesh, V., Meer, P. (2000). Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on* (Vol. 2, pp. 142-149). IEEE.

[30] Davis, J., Goadrich, M. (2006, June). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning* (pp. 233-240). ACM.

[31] Zdenek Kalal, Krystian Mikolajczyk, Jiri Matas(2010). Tracking Learning Detection. Poster session presented at *CVPR*.

[32] Zdenek Kalal (April 2011). Tracking Learning Detection (Doctoral dissertation, University of Surrey, 2011). Retrieved from http://www.ee.surrey.ac.uk/CVSSP/Publications/papers/Kalal-PhD_Thesis-2011.pdf

www.kth.se