NTNU – Trondheim
Norwegian University of
Science and Technology

Augmented Reality and Object Tracking for Mobile Devices

Eivind Gravdal

Master of Science in Engineering Cybernetics
Submission date: June 2012
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MSc. Thesis Assignment

Candidate's name: Eivind Gravdal

Course: Engineering Cybernetics

Thesis title: Augmented Reality and Object Tracking for Mobile Devices.

Assignment text:

In this project, the possibilities in Augmented Reality for mobile devices will be explored. Augmented Reality is a technology, where computer-generated information is applied to video of the physical world in real time. This information should be reactive with the surroundings and should give an impression of following real objects on the device screen.

There is already developed a system for recognition of objects, but the system is currently not fast enough to be used for an Augmented Reality applications. For the application to be able to add information directly on the objects that are recognized, there is thus a need to follow the object's location in real time.

The assignment will include:

- a search for the methods that are available and beneficial for the application.
- an evaluation of the existing system, where execution time and performance is analyzed. It should be considered whether some components can be used further for object tracking.
- a documentation of the final developed system.
- an evaluation of system performance, where the runtime and memory usage are important.

Goal:

To develop an Android application, where arbitrary objects can be followed on the device screen, while graphical information follows the object in real time.

The assignment is given: January 16, 2012

Date for submission: June 11, 2012

Carried out for the Department of Engineering Cybernetics, NTNU

Supervisor: Tor Onshus

Sammendrag

I *Utvidet Virkelighet*, kan datagenerert informasjon bistå brukeren med 'usynlig' informasjon om virkelige objekter. I denne avhandlingen, vil mulighetene for utvidet virkelighet på mobile enheter bli utforsket, og *objekt følging* er identifisert som en viktig funksjonalitet. I de eksisterende løsningene, er *posisjon- og orientering-basert følging* mest vanlig, men dette prosjektet vil derimot gå nærmere inn på *syns-basert følging*, som er et bredt forskningsfelt innen *Datasyn*. *Objekt følgeren* som er valgt, er den prisvinnende algoritmen TLD, som sammen med det nylig lanserte FastCV biblioteket, utgjør funksjonaliteten i den endelig presenterte *Android* applikasjonen. Applikasjonen er testet med hensyn til minnebruk og prosesseringstid, og det er observert fra forsøkene, at detektormodulen i TLD dominerer prosesseringstiden. Uansett oppfører den endelige *Objekt følgeren* seg etter hensikten, og god ytelse oppnås ved å redusere størrelsen på de prosesserte bilderammene fra mobilens kamera.

For det videre arbeidet, kan prosesseringstiden bli redusert enda mer, muligens ved å lage lengre intervaller mellom hver kjøring med detektormodulen. Også muligheten for å lagre objektmodeller og bruke dem senere bør utforskes. Dette vil være nyttig i en *Utvidet Virkelighet* applikasjon, der objektene vanligvis er forhåndsbestemte.

Abstract

In *Augmented Reality*, computer generated information can assist the user with 'invisible' information about real world objects. In this thesis, the possibilities for Augmented Reality on mobile devices will be investigated, and *Object Tracking* is identified as a vital functionality. In the existing solutions, position- and orientation-based tracking is most common, however, this project will approach vision-based tracking, which is a wide research area in the field of *Computer Vision*. The chosen object tracker, is the prize-awarded algorithm TLD, which together with the recently released FastCV library, makes up the functionality in the final presented application. The application is tested with regard to memory and processing time, and it is observed from the experiments, that the Detector module in TLD dominates the processing time. Nonetheless, the final Object Tracker performs as intended and decent frame rates is obtained by reducing the size of the camera image frame.

For further work, the processing time could be reduced even more, possibly by making a longer interval between each run with the detector module. Also the possibility to save object models and use them later on should be explored. This would be useful in an Augmented Reality application, where the objects usually is predetermined.

Preface

This project is carried out for the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The project is represented by this thesis, submitted for the degree of Master of Science (MSc).

I would like to thank my supervisor, Tor Onshus, for guiding me through this project. I would also like to thank, Frank Jakobsen and Tor Ivar Eikaas at Cyberlab, for discussing technical challenges and for pushing me towards the final result.

Thanks to Zdenek Kalal and Georg Nebehay, for submitting their ideas to the open source community, and for answering my, more or less, annoying questions about their work.

Last, but not least, I will express gratitude to my family and my friends, who have supported me through my academic years. Could not have done this without you.

Til Glenn Martin og Charlotte. Veldig imponert over hvordan dere har kommet dere gjennom det siste året. Nå er det bare å se fremover :)

Til minnet om de som gikk bort så alt for unge 22. juli 2011.

Contents

List of Figures

LIST OF FIGURES

List of Tables

Terminology

CV Computer Vision; a research field aimed at methods for acquiring, processing, analysing, and understanding images.

AR Augmented Reality; Combines a view of the real world with computer generated graphics.

TLD Tracking-Learning-Detection; Long-term tracking algorithm, developed by Z. Kalal.

LK Lucas-Kanade; Short-term tracking algorithm developed by B. D. Lucas and T. Kanade.

NN Classifier Nearest Neighbor Classifier; Classification method for supervised learning.

JDK Java Development Kit; an SDK for Java

SDK Software Development Kit; a collection of software development tools for a certain platform or framework.

JNI Java Native Interface; a framework, which enables Java to make calls to native functionality, written in other languages, such as C, C++ and assembly.

NDK Native Development Kit; a SDK for developing native code on a virtual machine system.

ADT Android Development Tools; a plugin for the Eclipse IDE.

IDE Integrated Development Environment; a software application, which provides most of the functionality needed for software development.

1

Introduction

Augmented Reality is a technological concept which was introduced in 1968 [1], but has been given more attention the last few years. As there is being released mobile devices like smartphones and tablet computers, with increasing amount of computational power, the demand for more intelligent applications is growing. Augmented Reality can provide 'invisible' informationon to a preview(or see-through) screen viewing the real world, which means the technology can be utilized to assist the user in many real-life situations. These situations could be;

- **Navigation;** where hints and directions could be applied directly to the field of view. (find your car in the parking lot, navigate to the closest public bathroom, get directions for the nearby restaurants)
- **Visualisation of dynamic systems or structures;** computer generated models could show up on top of physical objects. (could show how a historical structure were built, an engine's dynamics in slow motion, future plans for a physical area.)
- **Medical surgery;** the doctor could be able to see directly inside the patient.
- **Operation of machinery;** the pilot could see through the cockpit walls and see relevant information directly.

While one would need head mounted diplays (HMD) for the most advance applications, this is not yet commercially available for mobile platforms[2]. This project, however,

will focus on applications that is realistic for the current mobile platforms and used for, but not limited to, commercial use.

1.1 Thesis Structure

This thesis, which documents the solution from the project, is divided into three parts, Background Information in Part I, Methodical Approach in Part II and System Evaluation in Part III. The parts is there to group the chapters in a logical structure. There are different approaches to make augmented reality applications, and this will be the outline of Chapter 2. The chapter will present some of the commercially available 'apps', before useful tools to create such software are described.

As the augmented reality is supposed to react with physical objects in the real world, the targeted device must obtain location information about these objects. To address this functionality, the possibilities in the field of computer vision is investigated, where Object Tracking is a central topic. The object tracker chosen is the TLD algorithm, which is described in Chapter 3. This algorithm got a lot of attention, when it was published in 2010/2011 by Z. Kalal et al. The algorithm was released as an open source project and the implementation in Matlab could be freely downloaded from the project's homepage.

Both Android and iOS were considered as target mobile platform, and Android was chosen due to convenience and its open source policy. General information and useful tools for developing Android applications can be found in Chapter 4

To develop this object tracker for the Android platform, there is a need for software modules to interact with hardware, such as camera, screen display and touch screen. This functionality and its communication with the TLD object tracker is documented in the Part II about the Methodical Approach. This part contains the motivation for the chosen software modules in Chapter 5, the system from the top abstraction layer in Chapter 6, and a more detailed decription of the implementation in Chapter 7.

In Part III, the system performance is analysed with regard to memory usage and run-time in chapter 8. The analyses is evaluated and discussed in Chapter 9, before the thesis is concluded in Chapter 10. Suggestions for further work can be found in Chapter 11.

1.2 Project objectives

Too sum up the preliminary objectives and the main goal of the project, the following list is based on the project description:

- **Objective 1:** do a search for existing solutions and methods available, and suitable for this application. (Part I)
- **Objective 2:** make an evaluation of the initial system from the preproject, and comparison to the state-of-art. (Part II)
- **Objective 3:** present a documentation of the final system; conceptual and implementation. (Part II)
- **Objective 4:** give an evaluation of the final system, where memory usage and run-time should be considered as the critical factors. (Part III)
- **Main goal:** develop an application, which can follow selected objects on the target mobile device's screen, while graphical information follows the object's location in real-time.

Part I

Background Information

2

Augmented Reality

Augmented reality is a wide field of technology, combining multiple research areas, where the goal is to merge the physical reality together with computer generated graphics. A definition of Augmented Reality can be found in a survey by Azuma[3]:

> This survey defines AR as systems that have the following three characteristics:
>
> 1. Combines real and virtual
> 2. Interactive in real time
> 3. Registered in 3-D

To make a computer able to do this, it needs to obtain knowledge about the real world, and this makes sensors a very important part of what Augmented Reality can be. Sensor technology is also a vast research field, but for the scope of this thesis, the focus will be on sensors in the current mobile smart phones. A thorough article on this can be found in [4], where the following sensors is described; Ambient light(ALS), Proximity Sensor, Global Positioning System(GPS), Accelerometer, Compass, Gyroscope and Back-illuminated sensor(BSI). In addition, they possesses a high resolution camera, most of them, have even two(front and back). These sensors can again be divided into *Position- and Orientation-Based* and *Vision-Based* sensors. See Figure 2 below.

In section 2.1, a selection of the currently used Augmented Reality applications for iOS and Android will be described, and categorized with respect to the sensors used.

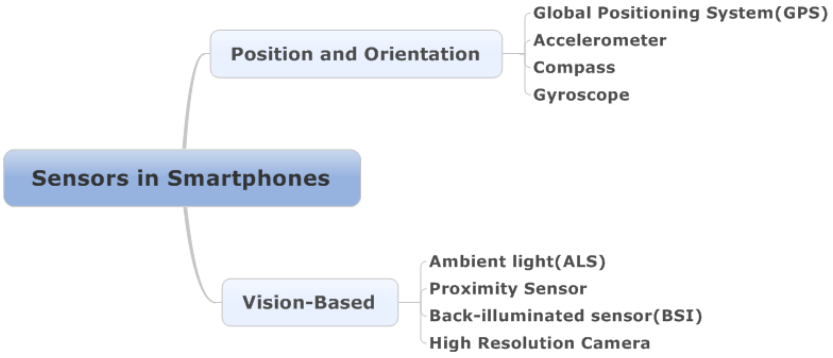

Figure 2.1: Sensors in Smartphones - Can be found in most of the recently released smartphones.

Last section in this chapter, Section 2.2, will discuss the common tools for making such applications.

2.1 Existing Solutions

2.1.1 Wikitude

Wikitude was released as a free Android application in 2008 by a Austrian development team from the company Mobilizy GmbH. This application gives the user information about points of interests in the real world directly on the screen merged with the camera preview. The user can point the mobile camera against the Statue of Liberty in New York, and Wikitude will display information from Wikipedia about this well known landmark. Wikitude is using position and orientation based sensors(GPS, accelerometer and compass), to show the information on the correct screen location.

2.1.2 Golfscape

Golfscape is, similarly to Wikitude, an Augmented Reality application based on tracking of position and orientation. This iOS application is developed by the American company Shotzoom LLC, and was released in July 2010 through the Apple App Store. The application displays useful information, like distance to hole, bunkers, water etc,

Figure 2.2: Wikitude - in front of Statue of Liberty. Picture is taken from [5].

and will display these distances in the appropriate direction on the camera preview screen.

Figure 2.3: Golfscape - Picture taken from [6].

2.1.3 Google Goggles

While Google Goggles actually is a image recognition application, it also contains a possibility for continous video detection. This feature sends multiple frames to the the recognition servers, and detects and tracks trivial information such as text, logos, landmarks or other user submitted items. Google Goggles performs vision based tracking, but it is only active for short amounts of time, as it stops when the target is recognized.

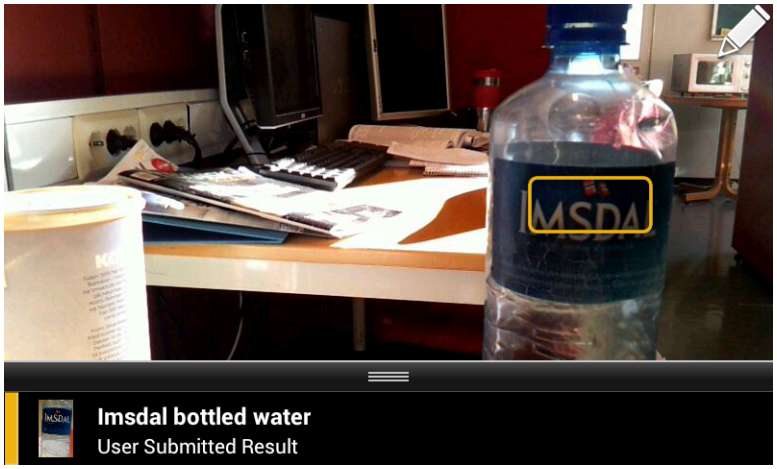

Figure 2.4: Google Goggles - A run in the office space.

2.2 Development Tools

Qualcomm has distributed a number of useful tools for developing mobile augmented reality applications. In the following subsections, two of these tools will be described. Subsection 2.2.1 reviews Qualcomm's Augmented Reality SDK *Vuforia*, while 2.2.2 is about their recently released Computer Vision library *FastCV*. The widely used Computer Vison library *OpenCV* is decribed in Subsection 2.2.3, and a small selection of other development tools, can be found in Subsection 2.2.4.

2.2.1 Vuforia

Vuforia, previously known as QCAR, is an SDK for development of AR applications on mobile devices, and is compatible with Android and iOS. Vuforia gives developers the possibility to use 2D and 3D markers, which serves as reference points in the real physical world. These markers is then tracked by the built-in tracking method, which makes it easy to apply animations or other computer generated graphic responding to the markers. The coordinate system will be centered at the marker, and rotated according to the marker's orientation in the 3D environment. The SDK was released in february 2012 in its new brand, as it was updated from QCAR SDK 1.0 to Vuforia SDK 1.5. The system architecture is as following; The developer provide target images to the *Target Management Application*, which loads the targets into the mobile application

as resources. The resources are used by the functionality in the QCAR Library, which again is called upon by the developer.

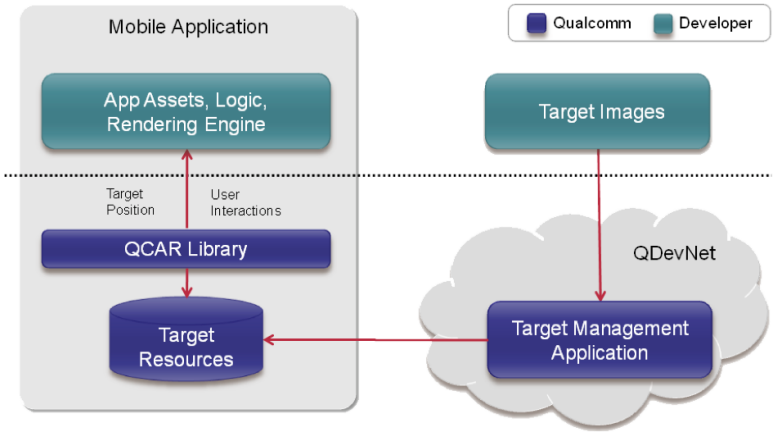

Figure 2.5: Vuforia - System Overview. Figure is from [7].

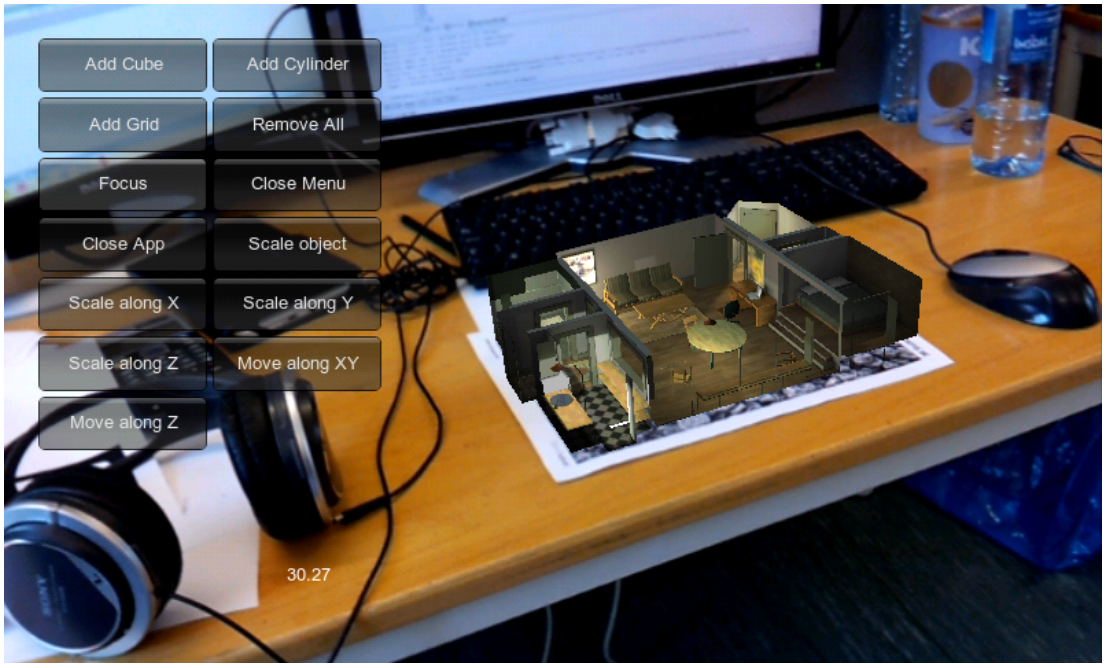

Figure 2.6: House model - This is from an application, made with the Vuforia SDK. The marker used can be found in the appendix, and the 'app' can be downloaded from `https://play.google.com/store/apps/details?id=org.monosock.shadowdemo&hl=en`.

Figure 2.7: House model - Close-up view of the model. The furnitures can be moved and doors can be closed from the touch screen.

2.2.2 FastCV

The computer vision library *FastCV* was released for public use late October 2011 [8]. This library has earlier been used by Qualcomm as the center piece in their QCAR SDK, but is now available for download at `https://developer.qualcomm.com/develop/mobile-technologies/computer-vision-fastcv`. The following quotation describes FastCV, from Qualcomm's point of view:

> "FastCV offers the most widely used, computationally intensive vision processing APIs, with hardware acceleration and better performance on mobile devices. It is designed for developers interested in creating sophisticated CV apps, as well as CV middleware developers looking to build the frameworks necessary for everyday developers to include computer vision functionality in their apps. FastCV is the framework at the heart of our vision-based Augmented Reality (AR) SDK, because AR is much more precise and useful when its based on camera input than on location-based estimates."

> *- Sayeed Choudhury, director of product management in Qualcomm*

The functionality in the FastCV[9] library can be divided into the following categories:

- Image Processing
- Image Transformation
- Feature Detection
- Object detection
- 3D reconstruction
- Color conversion
- Clustering and search
- Memory Management

2.2.3 OpenCV

OpenCV was originally developed by researchers in Intel, and was released to the public in 2000 at the IEEE Conference on Computer Vision and Pattern Recognition. OpenCV is a programmer library aimed at the computer vision community. The library contains functions to support real time imaging and other vision related applications. Since 2008, OpenCV is again actively being developed and maintained by the robotics research lab Willow Garage, which also maintains ROS (Robot Operating System) and PCL (Point Cloud Library). The vast amount of functionality in OpenCV[10], can be categorized as following:

- Image Processing
- High-level GUI and Media I/O
- Video Analysis
- Camera Calibration and 3D Reconstruction
- 2D Features Framework

- Object Detection
- Machine Learning
- Clustering and Search in Multi-Dimensional Spaces
- GPU-accelerated Computer Vision
- Computational Photography
- Images stitching

OpenCV is compatible with Windows, Mac OS, Linux and there is recently released versions suited for iOS and Android. The latest version 2.4 was released the 28th of April 2012, however version 2.3.1 for Android is used in this project.

2.2.4 Others

Following is a selection of other tools for developing AR applications:

- **ARToolKit** - is a library of augmented reality functions, which includes functionality such as marker tracking and graphical overlay with OpenGL. This has been one of the most used tools for developing AR applications since the realease in 1999. ARToolKit is available for Windows, Mac OS X and Linux, and has recently been ported to mobile platforms (Symbian, iPhone(2008)[11], Android and Windows phone). A number of extended versions and ports has also been developed by third party development teams.
- **PTAM** - or Parallel Tracking and Mapping for Small AR Workspaces [12] is a method for mapping the environments, as an alternative solution to problem adressed by SLAM(Simultaneous localization and mapping)[13]. PTAM can map the environment by estimating the camera pose from an unknown 3D scenery, which is computed from tracked keypoints. The algorithm is not meant for large scale mapping, but runs with the camera frame rate in a smaller office space.
- **Metaio** - is a distributor of Augmented Reality SDKs(Software Development Kits). Their mobile SDK supports Android and iOS, and its full version is free to download, in contrast to the other Metaio SDKs. The applications, however, will get a Metaio watermark and splashscreen, if not paid for.

3

Video Tracking with TLD

The TLD tracking algorithm [14] [15] was developed by Z. Kalal during his PhD thesis at the University of Surrey. In 2011, he was awarded the ICT Pioneers Prize in the 'Technology Everywhere' category for his work with this algorithm[16]. The algorithm combines elements from tracking, learning and detection (TLD), to calculate the scale and location of any selected object in the 2D image space. To make this a long-term tracker, these elements is correcting each other; the tracking module feeds the learning module with new data for a object model, while the detector module is using the learned data to correct or reset the tracker. This way, it will be stable in the long run, even though the object is disappearing, moving to fast, gets occluded or changes appearance.

Algorithm 1 and Figure 3.1 shows the different components in the tracker, which will be explained in detail throughout this chapter.

Input: Previous image frame I_{i-1}
Input: Current image frame I_i
Input: Previous bounding box BB_{i-1}

Output: Current bounding box BB_i
while *active input feed from camera* **do**
  Tracking
  - Forward-Backward tracking
    - Run Lucas-Kanade in two directions
  - Compute median flow
  Detection
  - Variance Filter
  - Ensemble Classifier
  - Nearest Neighbor Classifier
  Learning
  - P-N Experts
  Integrator
  - Validation
  - Evaluate BB candidates
end

Algorithm 1: The TLD tracking algorithm

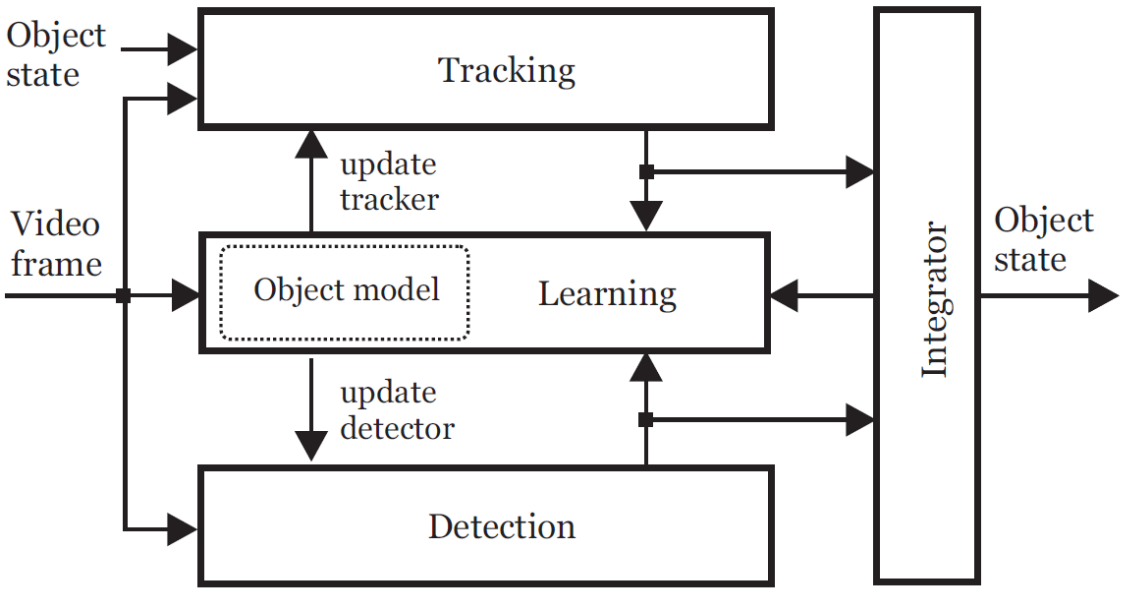

Figure 3.1: TLD Overview - Diagram of the main components in TLD. 'Object state' is the location and scale of the bounding box when object is tracked by TLD, and equals 'not visible' when TLD can't identify the object. Figure is taken from [14].

3.1 Tracking: Median Flow

The tracking solution is based on a previously released paper[15] by Kalal et al., and is also extended with a failure detector. The tracking is performed on an input bounding box, where the underlying patch is tracked from one camera frame and to the next. This is done by computing the optical flow(pixel displacement from one frame to the next) with the Lucas-Kanade(LK) algorithm on the pixels belonging to the patch(a 10 x 10 uniform grid of pixels is extracted from the patch). To ensure robustness, the LK algorithm is performed in two directions (first frame $\rightarrow$ second frame && second frame $\rightarrow$ first frame), and a reliability measure is computed for each displacement. The median displacement of 50 % of the most reliable displacements is considered as the object's motion from the first frame to the second frame. The Lucas-Kanade optical flow calculations is decribed in the following subsection.

3.1.1 Lucas-Kanade

The Lucas-Kanade algorithm was proposed at the International Joint Conference of Artificial Intelligence in 1981 [17], and has been widely used in the computer vision community since then. This method is used to compute the image alignment between a template image $T(\boldsymbol{x})$ and an input image $I(\boldsymbol{x})$, as explained in [18]. To compute the optical flow or to track the template image $T(\boldsymbol{x})$ from time t = 1 in the input image $I(\boldsymbol{x})$ from t = 2, the Lucas-Kanade algorithm calculates a warp matrix $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})$. $\boldsymbol{x} = (x, y)^T$ is a column vector describing the pixel coordinates and $\boldsymbol{p} = (p_1, \cdots, p_n)^T$ is a vector of parameters. For the 2D transformation, which is the case in optical flow, the warp matrix $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})$ can be equivalent to the translations:

$$\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}) = \begin{pmatrix} x + p_1 \\ y + p_2 \end{pmatrix} \tag{3.1}$$

Here $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ can be considered as the optical flow along x and y respectively.

To find $\boldsymbol{p} = (\boldsymbol{p_1}, \boldsymbol{p_2})^T$, the Lucas-Kanade algorithm will minimize the sum of squared difference between the template T and the image I:

$$\sum_x [I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})) - T(\boldsymbol{x})]^2 \tag{3.2}$$

To optimize the computation, it is assumed that the current estimate of $\boldsymbol{p}$ is known, hence it only needs to compute the incremental step $\boldsymbol{\Delta p}$ which is searched for in a local region with a Gauss-Newton approximation. Then the following expression is the subject for minimization:

$$\sum_{x}[I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}+\Delta\boldsymbol{p}))-T(\boldsymbol{x})]^2 \tag{3.3}$$

The current parameters $\boldsymbol{p}$ is updated, and the Gauss-Newton approximation can be performed with the new parameters.

$$\boldsymbol{p} \leftarrow \boldsymbol{p}+\Delta\boldsymbol{p} \tag{3.4}$$

These two steps is iterated until the estimated $\boldsymbol{p}$ converge.The test for convergence is typically, to check if the norm of $\boldsymbol{p}$ is below a certain threshold ϵ.

$$\|\Delta\boldsymbol{p}\| \leq \epsilon \tag{3.5}$$

To derive the solution of the non linear minimization problem from Equation 3.3, the expression is linearized by applying a first order Taylor expansion to I ($\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}+\Delta\boldsymbol{p})$):

$$\sum_{x}[I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))+\nabla I\frac{\partial\boldsymbol{W}}{\partial\boldsymbol{p}}\Delta\boldsymbol{p}-T(\boldsymbol{x})]^2 \tag{3.6}$$

The minimization problem can then be solved in the least square sense, where the partial derivative of Equation 3.6 is:

$$2\sum_{x}\left[\nabla I\frac{\partial\boldsymbol{W}}{\partial\boldsymbol{p}}\right]^T\left[I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))+\nabla I\frac{\partial\boldsymbol{W}}{\partial\boldsymbol{p}}\Delta\boldsymbol{p}-T(\boldsymbol{x})\right]^2 \tag{3.7}$$

This expression is equal to zero in the minimum point, and can then be solved with respect to $\Delta\boldsymbol{p}$:

$$\Delta\boldsymbol{p}=H^{-1}\sum_{\boldsymbol{x}}\left[\nabla I\frac{\partial\boldsymbol{W}}{\partial\boldsymbol{p}}\right]^T[T(\boldsymbol{x})-I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))] \tag{3.8}$$

where H is the $n\times n$ Hessian matrix:

$$H=\sum_{\boldsymbol{x}}\left[\nabla I\frac{\partial\boldsymbol{W}}{\partial\boldsymbol{p}}\right]^T\left[\nabla I\frac{\partial\boldsymbol{W}}{\partial\boldsymbol{p}}\right] \tag{3.9}$$

Input: Previous image frame I_{i-1}
Input: Previous bounding box BB_{i-1}
Input: Current image frame I_i

Output: Current bounding box BB_i
Output: Confidence map $conf[]$
while $\Delta p > \epsilon$ **do**
(1) Warp I with $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})$ to compute $I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))$
(2) Compute the error image $T(\boldsymbol{x}) - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))$
(3) Warp the gradient ∇I with $\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p})$
(4) Evaluate Jacobian $\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{p}}$ at $(\boldsymbol{x};\boldsymbol{p})$
(5) Compute the steepest descent images $\nabla I \frac{\partial \boldsymbol{W}}{\partial \boldsymbol{p}}$
(6) Compute the Hessian matrix using Equation 3.9
(7) Compute $\sum_{\boldsymbol{x}} [\nabla I \frac{\partial \boldsymbol{W}}{\partial \boldsymbol{p}}]^T [T(\boldsymbol{x}) - I(\boldsymbol{W}(\boldsymbol{x};\boldsymbol{p}))]$
(8) Compute $\Delta \boldsymbol{p}$ using Equation 3.8
(9) Update the parameters $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta \boldsymbol{p}$
end

Algorithm 2: The Lucas-Kanade algorithm, as in [18]

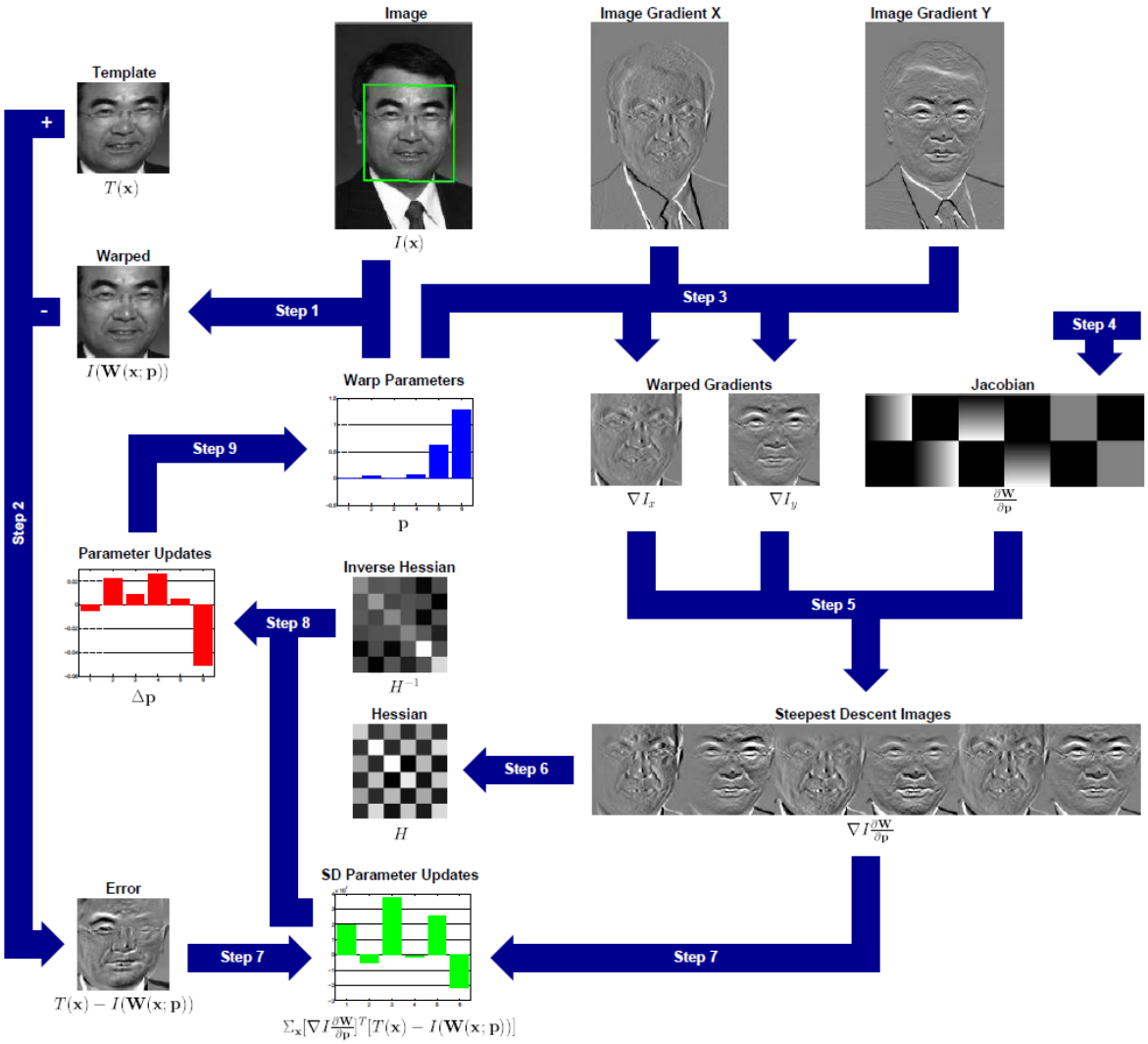

Figure 3.2: Lucas-Kanade Illustration - The figure follows the steps in Algorithm 2. Figure is taken from [18].

3.2 Detection: Cascaded Classifier

The Median Flow tracker assumes that the object will stay inside the screen view, and will fail if the object moves outside or is occluded. From the nature of the Lucas-Kanade algorithm it will also fail if the object moves to far from one frame to the next. To enable long-term tracking also in these cases, a detector is needed to search for the object when the tracker is lost.

The detector searches through the input frames with a scanning-window approach, and decides for each window's underlying patch if the object is visible or not. This scan generates a number of bounding boxes which needs to be evaluated with a confidence measure. This evaluation is done by a *Cascaded Classifier*, which can be divided into 3 stages:

1. Patch Variance Filter
2. Ensemble Classifier
3. Nearest Neighbor Classifier

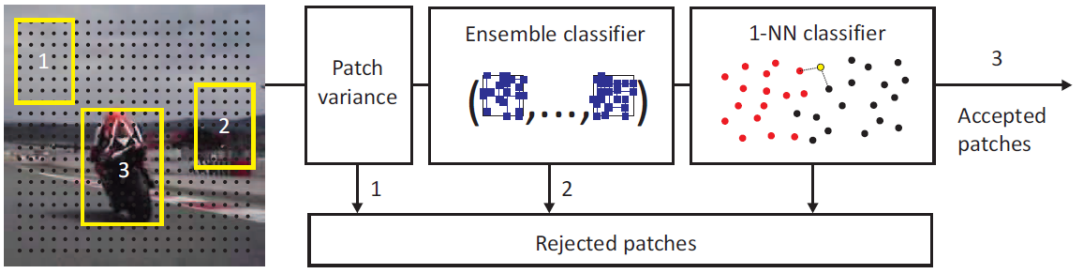

Figure 3.3: The detector used in TLD - Figure is taken from [14].

Eatch patch is sent through these stages, and is either rejected or passed on to the next stage. Following is a short description of the three stages:

3.2.1 Patch Variance

In this phase, the variance across the patch is computed. This is done by computing the mean with integral images, and then the variance is computed with the formula:

$$\sigma = m^2 - \frac{1}{N}\sum x^2 \tag{3.10}$$

where σ is the standard deviation, m is the mean, x is the pixel value and N is the number of pixels inside the patch.

The result is then compared to the initial patch of the selected object. If the variance is less than 50 % of the initial patch, the current patch is rejected. The patches with low variance is typically not containing objects (e.g. the sky, part of a wall). This process removes about 50 % of the patches in the original implementation, but a threshold can be adjusted to change the amount of patches passing by.

3.2.2 Ensemble Classifier

The ensemble classifier consists of n base classifiers, and each classifier i computes a posterior probability $P_i(y|x)$. This posterior probability is calculated from a number of pixel comparisons on the patch which is translated to a binary code. (see Figure 3.4) The setup of pixel comparisons is generated offline and each base classifier is designed to be independent of each other. In the original implementation there is generated 13 comparisons for each base classifier, and they will in total cover every pixel inside the patch.

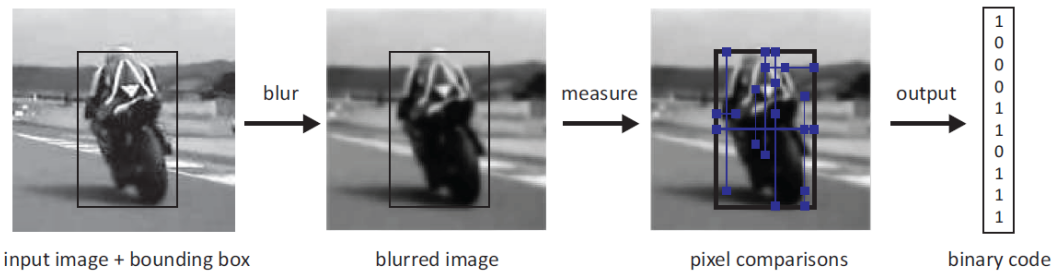

Figure 3.4: Conversion from patch to binary code - The input images is blurred by a Gaussian kernel to make the pixel comparisons robust against shift and image noise. Figure is taken from [14].

The posterior probability $P_i(y|x)$ is then equal to the number of positive patches with the same binary code as patch i, divided by the total number of negative and positive patches (related to the learning phase explained in Section 3.3).

$$P_i(y|x) = \frac{\#p}{\#p + \#n} \tag{3.11}$$

3.2.3 Nearest Neighbor Classifier

Finally, the resulting bounding box is chosen from the remaining patches with a Nearest Neighbor Classifier. In the original implementation there is typically 50 patches left after the two first stages of the Cascaded Classifier. To compare the patches to the negative and positive sets in the object model $M = (p_1^+, p_2^+, ..., p_m^+, p_1^-, p_2^+, ..., p_n^+)$, a number of similarity measures must be taken:

1. similarity with the positive nearest neighbor: $S^+(p, M) = max_{p_i^+ \in M} S(p, p_i^+)$
2. similarity with the negative nearest neighbor: $S^-(p, M) = max_{p_i^- \in M} S(p, p_i^-)$

where the similarity between two patches is

$$S(p_i, p_j) = 0.5(NCC(p_i, p_j) + 1) \tag{3.12}$$

and NCC is the $Normalized\ Correlation\ Coefficient$.

The patch is classified as the object if the relative similarity $S^r(p, M)$ exceeds an adjustable θ_{NN}:

$$S^r(p, M) > \theta_{NN}\ where\ S^r = \frac{S^+}{S^+ + S^-} \tag{3.13}$$

θ_{NN} can be tuned to change the balance between precision and recall (see Section 3.4).

3.3 Learning: P-N Learning

The learning module maintains the base of positive and negative patches in the following manner; first, a set of patches is initialized with respect to the user specified bounding box. Patches inside the bounding box is assigned as positive and the surrounding patches is assigned as negative. Then, during the online progress, the sets of negative and positive patches is updated according to socalled P-N-experts. These experts is described subsequently.

3.3.1 P-expert

The P-expert is supposed to detect new appearances of the object, and adds the corresponding patch to the positive set in the object model M. To locate these patches, the P-expert is exploiting the fact that the object moves along a trajectory. However, because the TLD algorithm is using both tracker, detector and an integrator to estimate the trajectory, the trajectory will be discontinous and not always correct. The task for the P-expert is then to find the reliable parts of the trajectory. The first step to find the reliable trajectory is to compute the *Conservative Similarity* S^c with the following expression:

$$S^c = \frac{S^+_{50\%}}{S^+_{50\%} + S^-} \tag{3.14}$$

where $S^+_{50\%}$ is the similarity between the patch and 50 % of the first registered positve patches $(p^+_1, p^+_2, ..., p^+_{m/2})$.

If S^c is larger than a certain threshold, the patch is added to the socalled *core*. The core is used to identify reliable parts of the object trajectory, which is shown in Figure 3.5. The trajectory is considered as reliable when it enters the core, but will be marked as unreliable if it stays outside the core. For every input frame, the P-expert decides if the current location is reliable or not. If the location is reliable a number of patches is picked from the surroundings, and is sent to the Ensemble Classifier together with warped versions of the same patches. In the original implementation, 10 patches is selected and warped 10 times each. This result in 100 patches for the Ensemble Classifier.

3.3.2 N-expert

The task for the N-expert is to label the negative patches of the training set. It can be assumed that the object is only located at one place at any time, and the N-expert is exploiting this. Every patch that is more than a distance threshold from the object is labeled as negative. The N-expert does this update after the Variance Filter and the Ensemble Classifier, and is only considering the remaining patches after these stages.

Finally, the integrator combines the bounding box found by the tracker, and the one found with the detector. The confidence, measured by the Conservative Similarity

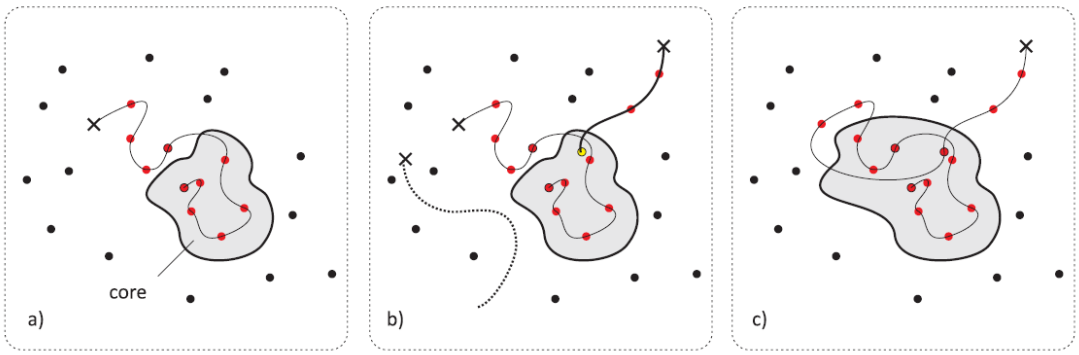

Figure 3.5: P-expert - Red dots are positive patches, black dots are negative. a) - The feature space with the core in shaded area. b) - unreliable and reliable trajectories, as dotted and continous lines respectively. c) - the object model and the core after update. Figure is taken from [14].

S^c, is evaluated for the two bounding boxes, and the box with the highest confidence is selected as the output bounding box. If there is not proposed a candidate bounding box from neither the detector nor the tracker, the object is considered as 'not visible'.

3.4 Performance: Precision/Recall

To measure the quality of the tracking, two entities is often used; *Precision* and *recall*. Precision describes the fraction of the selected keypoints belonging to the object divided by the total amount of selected keypoints. Recall is the fraction of the selected keypoints divided by the total number of keypoints belonging to the object. An example is shown in Figure 3.6. Sometimes, a combined measure of these two entities, called F-measure, is used:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.15}$$

In Figure 3.7 the precision and recall measures of TLD is compared against other state-of-the-art object trackers.

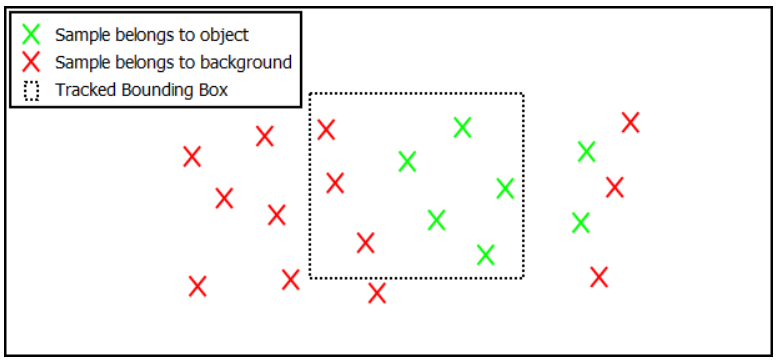

Figure 3.6: Example about precision and recall - Figure shows a number of sample points from a video frame. Green points belongs to the object, while red points belongs to the background. The bounding box is found by the tracker, and the performance on this frame is evaluated as follows; number of samples inside bounding box = 8, green samples inside = 5, total number of green samples = 7. Precision is then 5/8 = 0.63, Recall = 5/7 = 0.71

Sequence	Frames	OB [29]	SB [31]	BS [69]	MIL [30]	CoGD [33]	TLD
1. David	761	0.41 / 0.29 / 0.34	0.35 / 0.35 / 0.35	0.32 / 0.24 / 0.28	0.15 / 0.15 / 0.15	1.00 / 1.00 / 1.00	**1.00 / 1.00 / 1.00**
2. Jumping	313	0.47 / 0.05 / 0.09	0.25 / 0.13 / 0.17	0.17 / 0.14 / 0.15	1.00 / 1.00 / 1.00	1.00 / 0.99 / 1.00	**1.00 / 1.00 / 1.00**
3. Pedestrian 1	140	0.61 / 0.14 / 0.23	0.48 / 0.33 / 0.39	0.29 / 0.10 / 0.15	0.69 / 0.69 / 0.69	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
4. Pedestrian 2	338	0.77 / 0.12 / 0.21	0.85 / 0.71 / 0.77	1.00 / 0.02 / 0.04	0.10 / 0.12 / 0.11	0.72 / 0.92 / 0.81	**0.89 / 0.92 / 0.91**
5. Pedestrian 3	184	1.00 / 0.33 / 0.49	0.41 / 0.33 / 0.36	0.92 / 0.46 / 0.62	0.69 / 0.81 / 0.75	0.85 / 1.00 / 0.92	**0.99 / 1.00 / 0.99**
6. Car	945	0.94 / 0.59 / 0.73	1.00 / 0.67 / 0.80	0.99 / 0.56 / 0.72	0.23 / 0.25 / 0.24	**0.95 / 0.96 / 0.96**	0.92 / 0.97 / 0.94
7. Motocross	2665	0.33 / 0.00 / 0.01	0.13 / 0.03 / 0.05	0.14 / 0.00 / 0.00	0.05 / 0.02 / 0.03	0.93 / 0.30 / 0.45	**0.89 / 0.77 / 0.83**
8. Volkswagen	8576	0.39 / 0.02 / 0.04	0.04 / 0.04 / 0.04	0.02 / 0.01 / 0.01	0.42 / 0.04 / 0.07	0.79 / 0.06 / 0.11	**0.80 / 0.96 / 0.87**
9. Carchase	9928	0.79 / 0.03 / 0.06	0.80 / 0.04 / 0.09	0.52 / 0.12 / 0.19	0.62 / 0.04 / 0.07	0.95 / 0.04 / 0.08	**0.86 / 0.70 / 0.77**
10. Panda	3000	0.95 / 0.35 / 0.51	1.00 / 0.17 / 0.29	0.99 / 0.17 / 0.30	0.36 / 0.40 / 0.38	0.12 / 0.12 / 0.12	**0.58 / 0.63 / 0.60**
mean	26850	0.62 / 0.09 / 0.13	0.50 / 0.10 / 0.14	0.39 / 0.10 / 0.15	0.44 / 0.11 / 0.13	0.80 / 0.18 / 0.22	**0.82 / 0.81 / 0.81**

Figure 3.7: Performance of a selection of trackers - This table, presented in [14], shows TLD's performance(precision/recall/F-measure), compared to other trackers. The trackers is runned on different video sequences which is commonly used for testing these methods. The trackers in the test is; OB = Online Boosting[19], SB = Semi-Supervised On-line Boosting for Robust Tracking[20], BS = Beyond semi-supervised tracking[21], MIL = Multiple Instance Learning[22], CoGD = Co-trained Generative-Discriminative tracking[23]

4

Android

The Android platform were introduced to the market in 2007, and was pushed forward by the *Open Handset Alliance*, a collaboration between 86 hardware, software and telecom companies led by Google. The platform was launched to be an open mobile standard, and the architecture is based on the Linux kernel modified by the Android development team from Google. As of third quarter 2011, Android was estimated to have 52,5 % of the worldwide smartphone market [24], followed by Symbian and iOS.

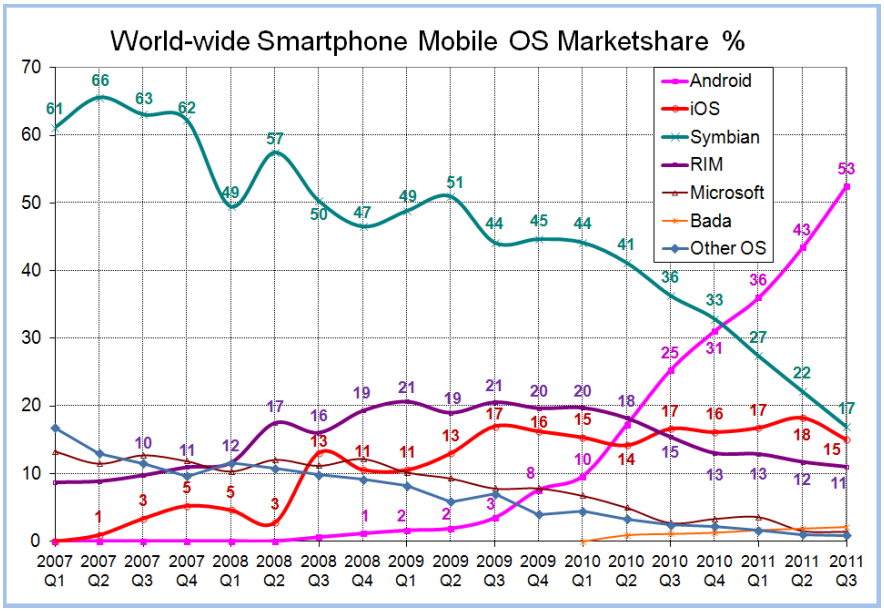

Figure 4.1: Smartphone Marketshare - graph over the last years from Gartners quarterly reports.

To start writing your own Android applications, a wide amount of useful information and tools can be found on the developers' homepage `developer.android.com`. The following sections will describe the tools and system configuration required to start programming, and it will be given an introduction to terms and important concepts.

4.1 Development Tools

To set up a development environment for the Android platform, a number of software components is required. The traditional and officially supported way to do this, is through the Eclipse environment. The following list (details in [25]) describes the necessary steps before starting an Android development project:

(1) Preparing Your Development Computer

- install Java Development Kit (JDK). (`http://java.sun.com/javase/downloads/index.jsp`)
- install a suitable version of Eclipse, 'classic' is recommended. (`http://www.eclipse.org/downloads/`)

(2) Downloading the SDK Starter Package

- install the latest version of Android SDK. (`http://developer.android.com/sdk/index.html`)

(3) Installing the ADT Plugin for Eclipse

(4) Adding Platforms and Other Packages

Additionally for native programming:

(5) Install Native Development Kit (NDK) [26].

- Download the appropriate package of the latest version of NDK. (`http://developer.android.com/sdk/ndk/index.html`)
- Extract archive into your development folder.

4.1.1 Android SDK

The Android Software Development Kit (SDK) was released in 2007 shortly after the Android platform was made public. The SDK was introduced to enable development of applications ('apps') to the new platform, and includes a debugger, software libraries, a handset emulator, sample code and tutorials. The Android SDK can be downloaded from `http://developer.android.com/sdk/index.html`, and is compatible with Windows, Mac OS X, and Linux.

4.1.2 Android NDK

The Native Development Kit(NDK) allows developers to write their code in native code(c/c++). This is not allways preferable, as development with the NDK always adds complexity to the system, but does not always increase performance. The NDK should be used when performing CPU intensive computations, such as signal processing, physics simulations or other algorithms doing large scale matrix calculations.

4.2 Terms and Concepts

The Android application is defined by the AndroidManifest.xml file, where the system stores which activities can be launched from the application, and which activity should be launched when the application starts up. The Android activities is used throughout the programming process to define separate events which the user can start. The running activity is maintained by the activity manager and is driven by the activity lifecycle described in Figure 4.2.

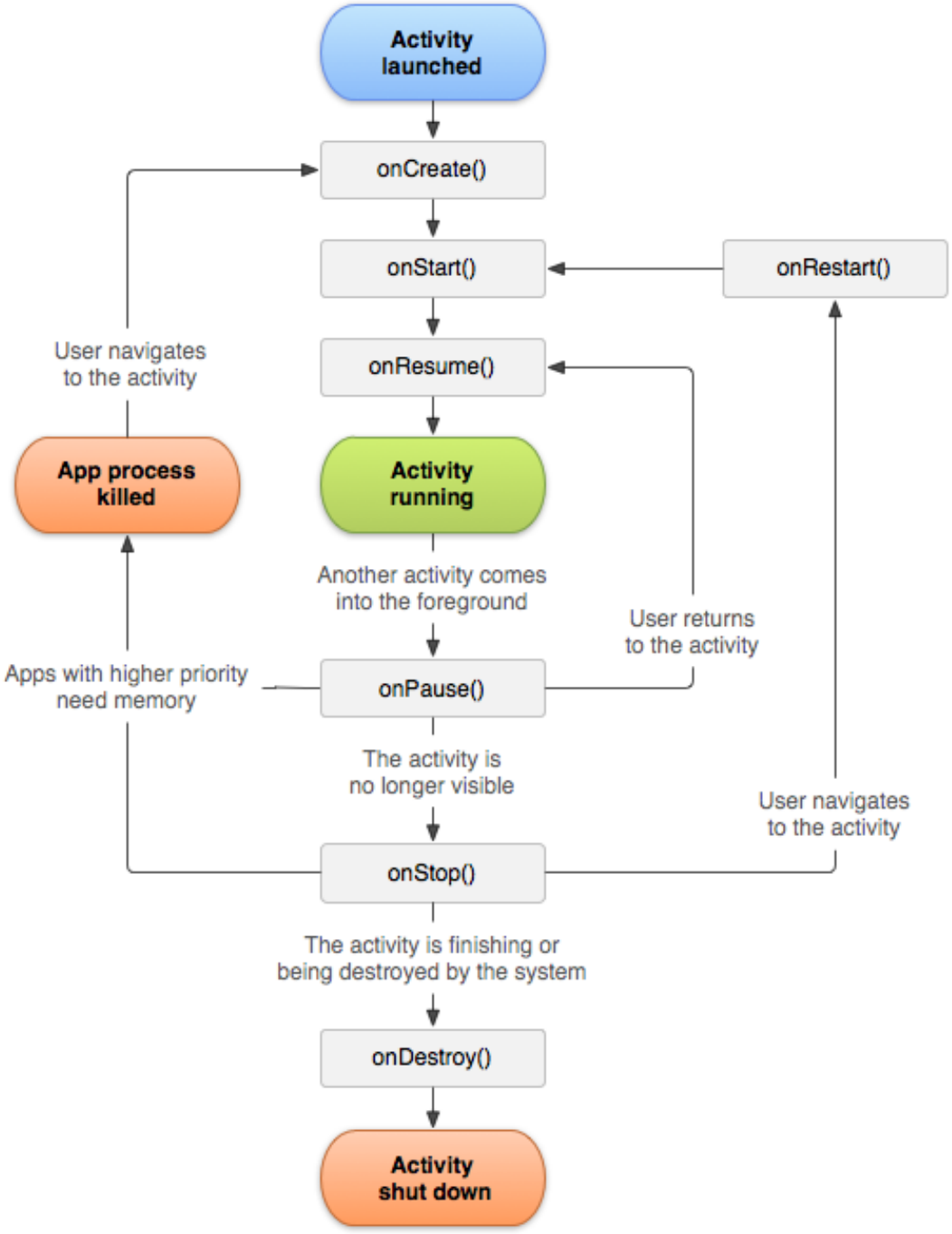

Figure 4.2: The activity lifecycle - The diagram describes the possible states of the activity, and which functions that trigger the next state.

Part II

Methodical Approach

5

Motivation

The main objective of this project was to make a long-term object tracker, to enable development of augmented reality applications. Several approaches were considered, and a large emphasis was laid on frame rate and robustness. Another factor which was kept in mind, was the limitations of the mobile platform, where the processing power and the memory available still is a significant gap from the regular work stations. This chapter describes the solutions chosen, to construct this long-term tracking application for the Android platform. The most important functionality can be grouped into three software modules:

1. **Image Acquisition** - should be able to to grab image frames from the camera, and convert to the appropriate image format.

2. **Object Tracking** - should be able to track an object's position and scale from the input image frame and the initial position of the object.

3. **Rendering** - should be able to construct the output image frame and bring it to the screen.

A selection of existing methods is evaluated in the following sections, where the motivation for choosing the object tracker and the two other modules is presented in Section 5.1 and Section 5.2 respectively.

5.1 Object Tracking

First of all, the video tracking problem is addressed. Obviously, this could be solved by performing object recognition (previous work in [27]) on every frame, but there is several refined methods which can do the job in a more elegant manner. The following methods were looked into: Rosten and Drummond's tracking[28][29] based on FAST feature points, Viola and Jones' object detection framework[30], Wagner's et al. approach[31][32] for tracking *natural features* on mobile phones, Ozuysal's et al. method[33][34] based on *random ferns* and Kalal's et al. 'Tracking-Learning-Detection' (TLD) algorithm[14] explained in chapter 3. All these methods can do object tracking with a reasonable frame rate, e.g compared to tracking with original SIFT or SURF features (i.e. to use highly invariant features for object recognition on every frame will be too computational expensive). Below, in Table 5.1, is the result from a test run with most of the supported feature detectors in OpenCV (8/10 detectors, 2 did not run properly). An alternative for a more thorough comparison, can be found in [35].

	FAST	STAR	SIFT	SURF	ORB	GFTT	Harris	DENSE
Computation Time(ms)	83	755	27577	5861	282	863	806	237
Number of Keypoints	2206	259	1228	1505	702	1000	502	10720
Theoretical fps(1/time)	12.05	1.33	0.036	0.17	3.55	1.16	1.24	4.22

Table 5.1: Test: OpenCV feature detectors - The test is performed on an arbitrary frame, and the detectors is runned sequently on the frame with the default threshold values. The frame size is 800 x 480, and the detectors is from OpenCV v2.3.1 for Android. The test code can be found in workThread.cpp on the attached CD.

From this test, OpenCV's FAST implementation has the lowest runtime, and could possibly be a good basis for a real-time tracker, as proposed in [28]. Ozuysal et al. presents a method where random ferns is used to recognize objects. The ferns is pixel comparisons put through a binary test, where the pixel locations is randomly selected in pairs to maintain statistical independence. A patch(subset of the frame) is then classified by posteriori probabilities calculated from the tested ferns. In [31] it is introduced a method which combines tracking of natural features using SIFT and fern detection. In the modified SIFT algorithm Wagner is using a FAST feature detector instead of the Difference of Gaussian(DoG) approach in SIFT[36], which greatly reduces

its computational cost.

The TLD approach, which was chosen as the solution for this project, takes some elements from the methods above. TLD is also using ferns for detection, machine learning plays an important role, and tracking is done with a modification of the widely used Lucas-Kanade algorithm. Several of the methods above could possibly be successfully implemented for Android, however, there was a few significant reasons to choose TLD;

1. TLD's claimed performance was convincing.
2. The project is open-source, so the method could easily be tested in Matlab.
3. An active open source community, where the solution was discussed and developers gathered for portation to C++.
4. A clean and effective C++ port(`https://github.com/gnebehay/OpenTLD`), by G. Nebehay.

The community gathered at `http://groups.google.com/group/opentld`, and a number of C++ ports were developed during fall 2011. In December, three ports were announced by alantrrs, aonsquared and gnebehay (Nicknames). Alantrrs' and aonsquared's ports were merged, but seemed to have some issues with the frame rate. As the speed was critical for this project, gnebehay's port was selected. This port, by G. Nebehay can be built with CMake, thus the compability is not an issue with most platforms. It was tested by the author on Ubuntu 11.10 and Windows 7, before starting the application development for Android. Below, in Table 5.2 is the frame rate measures, tested in [37].

	OpenTLD(C++) ST	OpenTLD(C++) MT	TLD (Matlab)
Total Computation Time(ms)	18390	14120	38170
Computation Time per frame(ms)	58.75	45.11	121.95
Frames per second	17.02	22.04	8.20

Table 5.2: Test: Comparison of OpenTLD and original TLD. - This is a comparison of the C++ implementation OpenTLD and the original Matlab implementation. The test is performed on a video sequence called 'Jumping', which consists of 313 frames. ST means single-threaded, and MT means multi-threaded. Data is taken from [37].

5.2 Image Acquisition & Rendering

To make the application work as intended, it was necessary to; 1. bring the input images from the Android camera and into the object tracker, and 2. bring the tracked bounding box onto the preview screen. After testing with different solutions, FastCV's sample application 'FAST Corner sample' stood out as the most promising framework that could take care of these tasks(1 & 2). This sample application was intentionally written by Qualcomm to demonstrate the performance of FastCV, and it is compared in Table 5.3 with the author's implementation of a feature detector using OpenCV.

	FastCV detector	OpenCV detector
Computation Time(ms)	14	46
Number of Keypoints	2361	2296
Theoretical fps(1/time)	71.43	12.05

Table 5.3: Test: Comparison of FastCV's sample application and the initial implementation using OpenCV - The OpenCV result is taken from Table 5.1, and the FastCV result is taken from an arbitrary frame during runtime(Selected to match the amount of keypoints in the OpenCV test). Both detectors is using the FAST algorithm for detection.

Following, in Figure 5.1 and Figure 5.4, follows a complete analysis of the two system's run-time. The implementation is divided into modules, mainly covering Image Aquisition, Object Detection and Rendering. Data conversion and decoding is also included due to it's significant amount of processing time. The remaining processing is registered as Computational Overhead and is grouped together with the Image Acqusition module, which is difficult to measure accurately. The processing time is computed from time intervals, defined by the `System.nanoTime()` function in Java(Interval = endTime - startTime), which measures the time elapsed since first of January 1970 in nanoseconds. The total processing time is computed with the same procedure, just by setting a time when one frame arrives and compare it to the time when the next arrives. To determine the time intervals in the C++ files, the function `getTimeMicroSeconds()` is used, which further calls on `clock_gettime(CLOCK_REALTIME, &t)`.

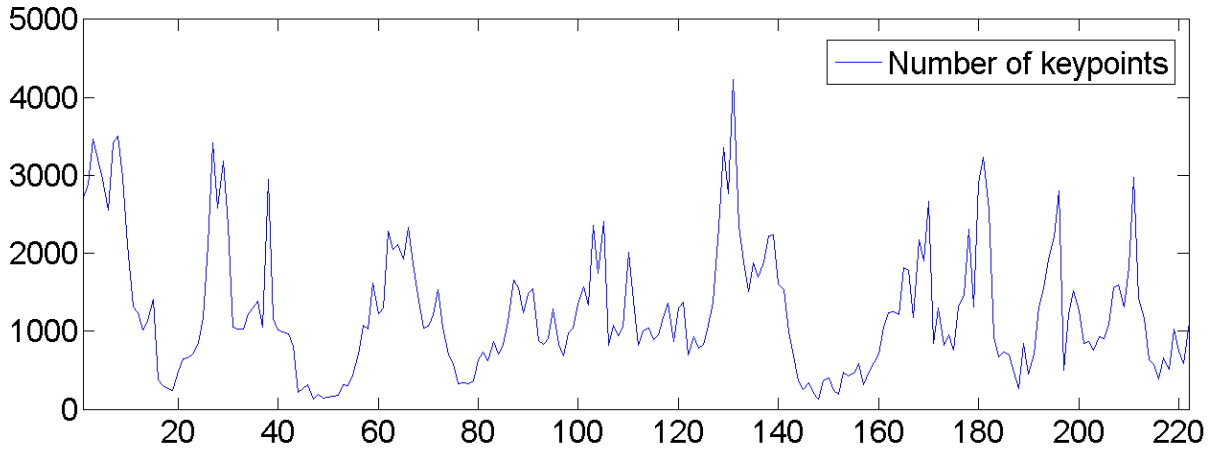

Figure 5.1: Initial implementation - Number of keypoints

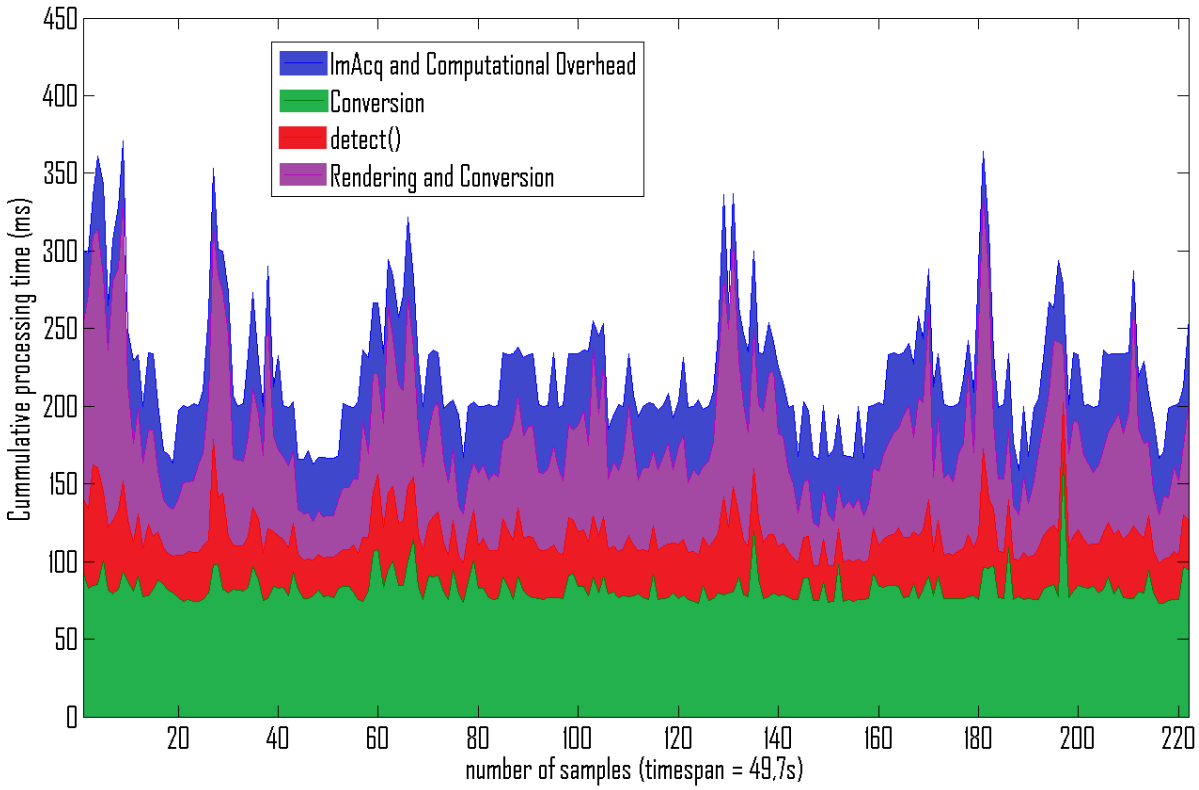

Figure 5.2: Initial implementation - Processing time analysis.

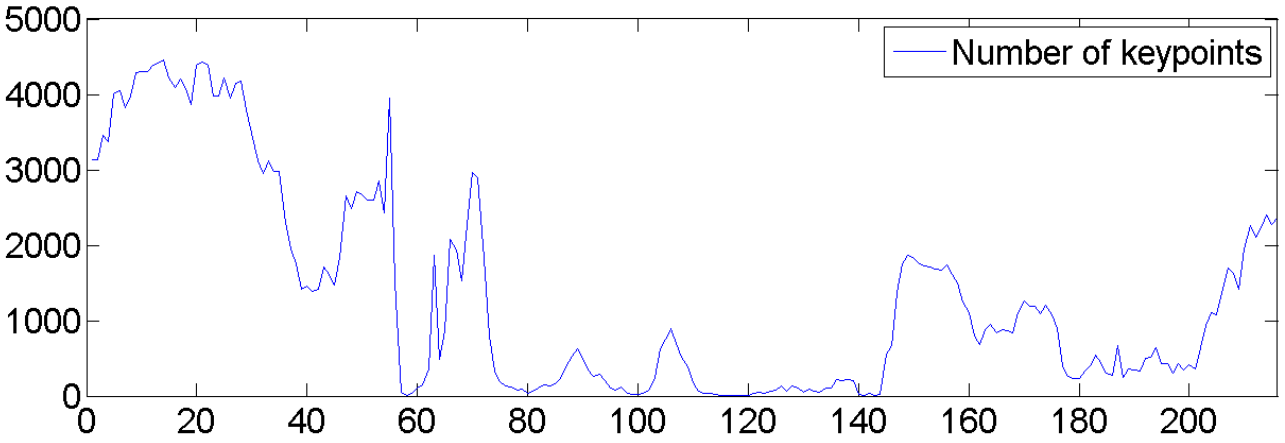

Figure 5.3: FastCV Sample - Number of keypoints

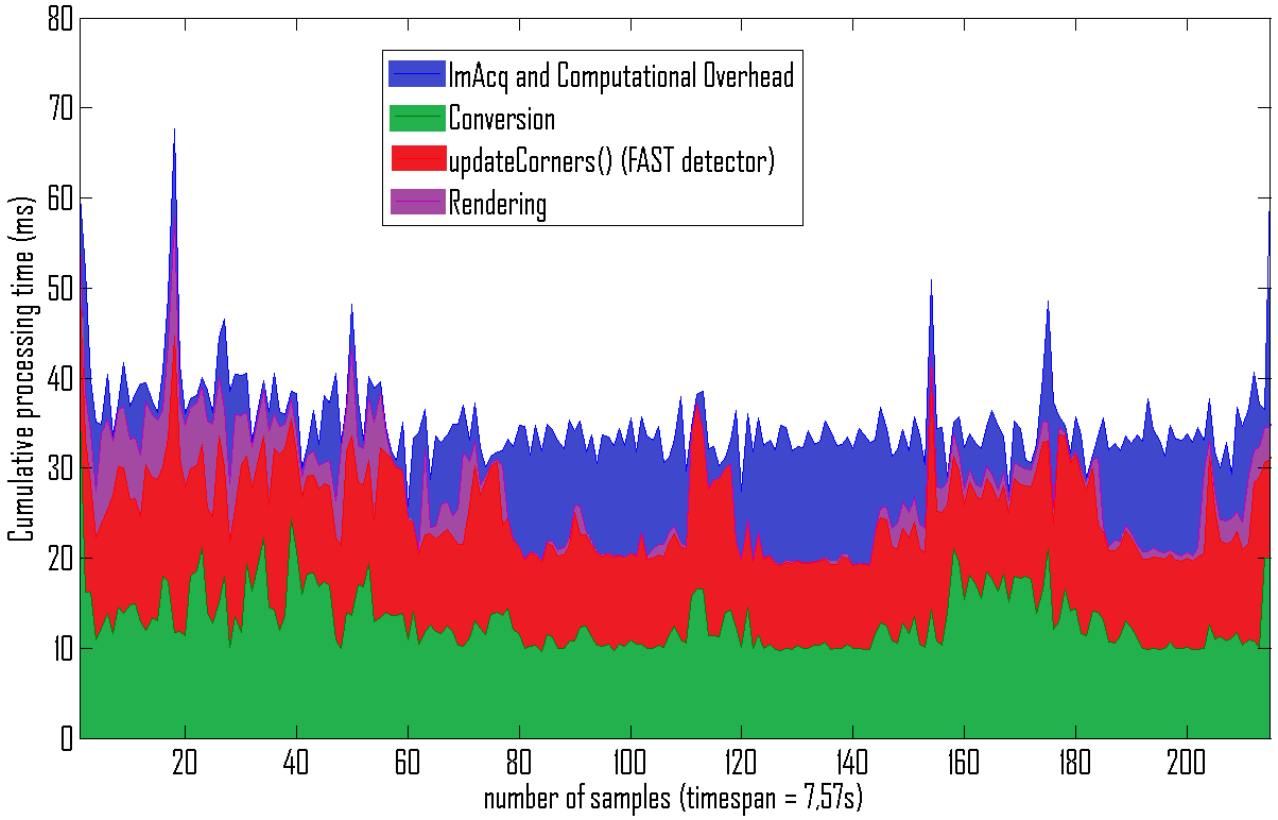

Figure 5.4: FastCV Sample - Processing time analysis.

From the run-time analysis it can be seen that the FastCV sample outperforms the initial implementation for each of the defined modules. In the initial implementation, the data conversions and rendering is written in Java, while the detector is called through JNI. In the FastCV sample, all computational expensive modules is processed through JNI, which seems advantageous. It also has an elegant rendering module, where changes in the output image is stored in a buffer, which is sent to the screen when it is ready to display a new frame.

6

Technical Overview

In Chapter 5, the motivation for choosing OpenTLD and the FastCV Sample as solutions was looked into. In this chapter, the structural overview of the final system will be described and visualized. In Section 6.1, the system is described with respect to the interaction with the user. In Section 6.2, an overview of the functionality is given from the programmers point of view.

6.1 User perspective

The progress in the application is defined by only a few events from the user perspective. The following diagram, in Figure 6.1, shows these events through a typical run of the application, from the user's point of view:

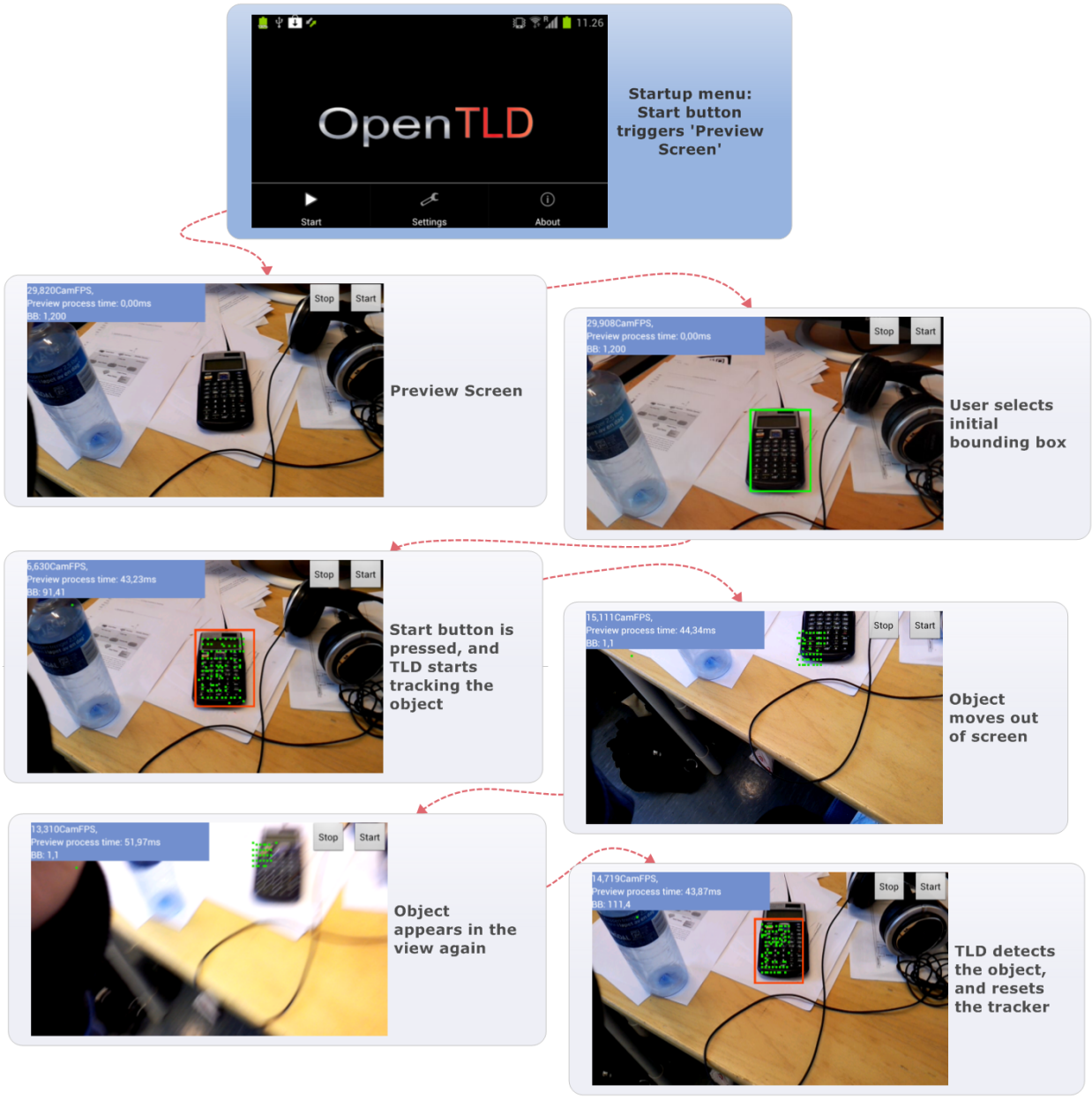

Figure 6.1: Application from the user's point of view - From the opening screen, the user has 3 possibilities, 'Start', 'Settings', 'About'. The 'About' view contains a short summary about the application, 'Settings' contains adjustable variables(such as image scale, thresholds, on/off modules etc.), and 'Start' triggers the preview screen. The user then selects the object from the current freezed preview with the touch screen. TLD starts after the second 'start' button is pushed, and the user does not need to interact with the system again, unless a new object is to be selected. This example used scale 4 image frames.

6.2 Programmer perspective

This section contains two diagrams giving an overview of the software. The first diagram in Figure 6.2, separates the software from the hardware of the targeted mobile device and describes the purpose of the most important Software modules. As elaborated in Chapter 5, the object tracking solution is performed with the OpenTLD implementation by G. Nebehay, while Image Acquisition and Rendering is taken care of by a framework based on the FastCV sample application. The second diagram describes the main modules of the OpenTLD implementation and how they work together.

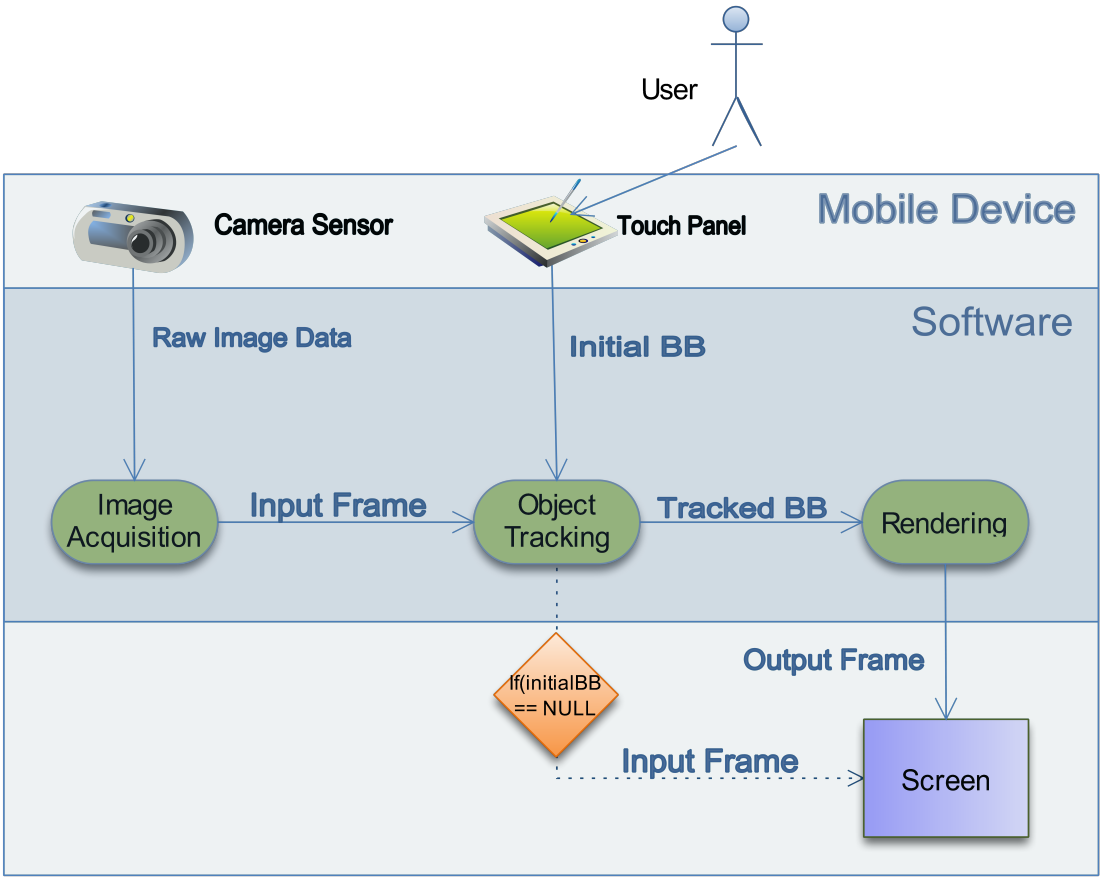

Figure 6.2: System overview - This figure shows the main components of the system. The image acquisition module takes the image data from the sensor, and sends it as an appropriate datatype to the object tracking module. When the object tracker receives the initial bounding box from the touch panel, it will compute the tracked bounding box in the next input frame. The input frame and the tracked bounding box is sent to the rendering module, which produces the output frame. If no initial bounding box has been set by the user, the output frame will equal the input frame

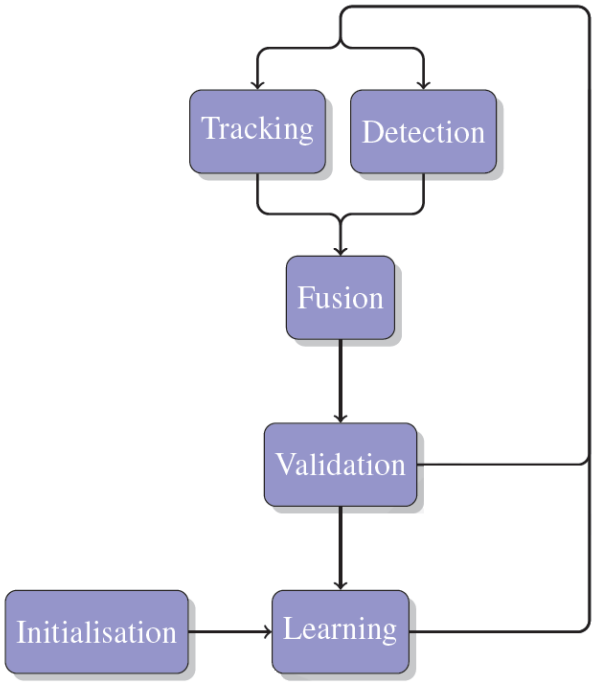

Figure 6.3: OpenTLD overview - The TLD object tracker is launched when the user selects a bounding box and clicks start on the preview screen. When this happens, the TLD Initialisation step is started, which sends the initial input frame together with the initial bounding box, into the Learning module. This step produces the first positive patch, which is subject for comparison in the Detection module. Next, the Detection module and the Tracking module is runned independently, before the Fusion module evaluates the two results. A validation step is then deciding whether the patch, representing the tracked bounding box, should be subject for learning or not. Diagram is from G. Nebehay's Master thesis(draft version)[37].

7

Implementation

In Chapter 6, the system is described from the view of the user, and then from the top level for the programmer with regard to the software modules used in the implementation. In this chapter, the implementation will be explained from a lower abstraction layer. The application flow will be described with sequence diagrams, and relevant function calls will be pointed out (arguments can be seen in source code, but is mostly marked with '...' in functionCalls(...)). It is recommended to use this documentation, when going through the source code for the first time. Section 7.1 walks through the application flow in the framework used, which is based on the FastCV sample application. Section 7.2 looks at the flow through the object tracker, which is the C++ implementation of OpenTLD, by G. Nebehay.

7.1 Application framework

When the application is launched, the SplashScreen activity starts up as defined in the OpenTLD manifest file AndroidManifest.xml. The FastCVSample activity(see Figure 7.1) is then started in SplashScreen.java when the menu start button is pressed by the user, and begins with the `onCreate()` call. The next step is the `onResume()` call where the initialisation of the Java part takes place. 'Start' and 'Stop' buttons is defined here, preview screen is activated, and there is a `onTouch(...)` listener as well.

When the user touches the screen, it should freeze the current preview frame, and a green bounding box appears where the user touched the screen. The bounding box can be dragged around and resized to the desired shape and size. This is done by registering different touch events with the `MotionEvent` functionality such as `ACTION_DOWN`, `ACTION_MOVE` and `ACTION_UP`. The first touch is registered as a fixed corner, and the second corner follows when the touch moves across the screen. If an object is selected, the user can press 'Start' to start tracking with TLD. The `onTouch(...)` listener is still active during tracking, which means that the user can reselect object from the preview screen at any time.

When 'Start' button is pressed, it sends a call to the JNI function `initTLD(...)`, where the selected bounding box and the freezed image is sent to the object tracker module with the function `selectObject(...)`. This step is initialising variables used by the tracker. For every frame from the preview screen the function `onPreviewFrame(...)` is processed, and calls the JNI function `update(...)` in FastCVSample.cpp. After Initialisation, this module launches the OpenTLD implementation with a call to the function `processImage(Mat grey)`. The `update(...)` function is also converting the raw image data to the OpenCV matrix format Mat, before it is sent in grayscale to the object tracker. When the OpenTLD module has computed the tracked bounding box, it is made available for the Rendering module and drawed onto the output buffer frame with the function `drawBB(...)` in FastCVSample.cpp. As soon as the screen is ready to display another frame, the buffer is rendered on the screen by the `requestRender()` function in FastCVSample.java.

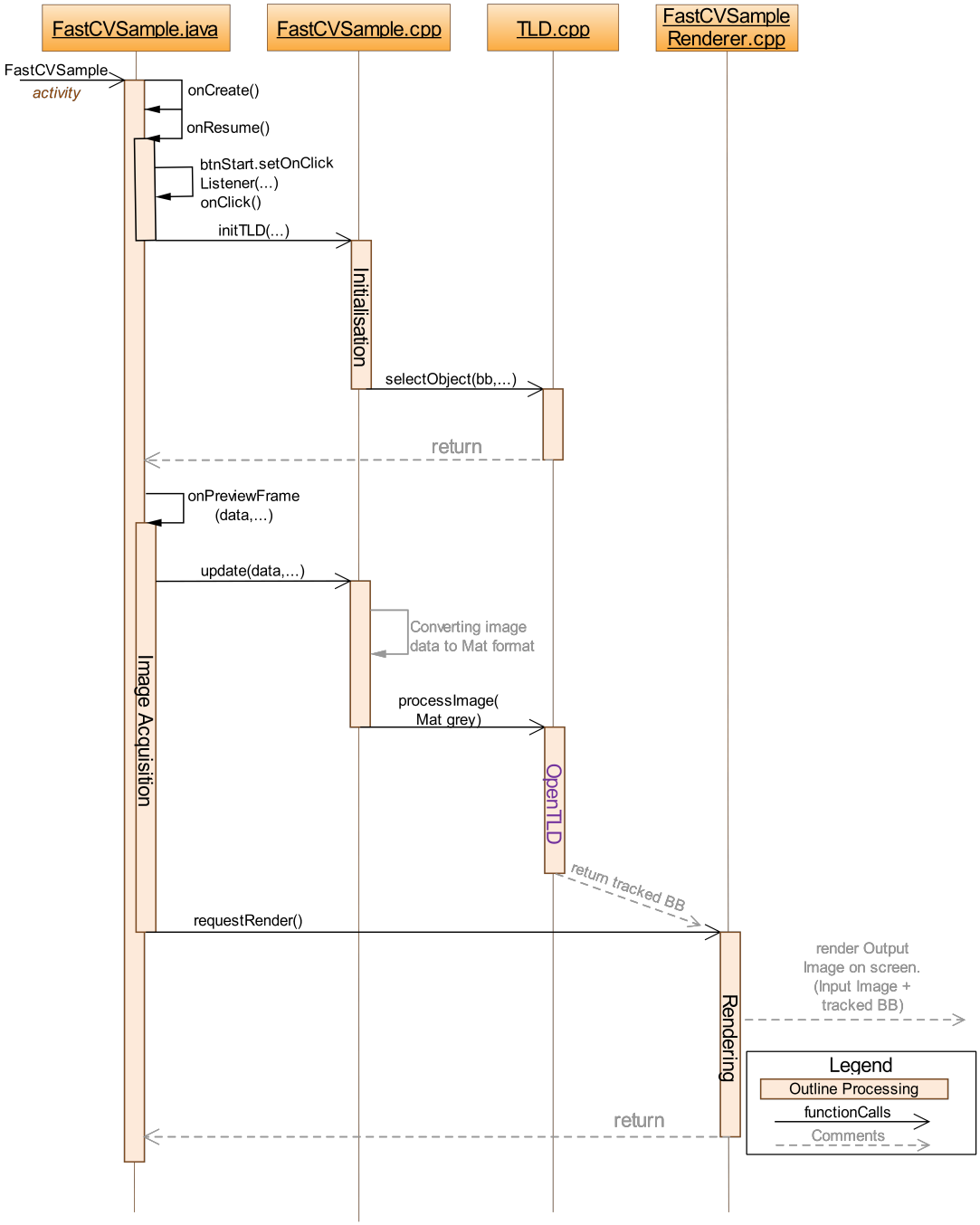

Figure 7.1: Implementation: Application framework - FastCVSample activity is launched by SplashScreen.java and starts with the onCreate call(see the Activity lifecycle in Section 4.2) when the menu start button is pressed by the user. To communicate between the Java and C++ files, the Java native interface(JNI) is used, and the C++ files is compiled with the Android NDK 4.1.2

7.2 Object Tracker

The object tracker is trigged from the file FastCVSample.cpp with a call to the function `processImage(Mat grey)` in `TLD.cpp`. This function operates the modules; Tracking, Learning and Detection(see Figure 7.2. The Tracking module is launched with the `track(...)` function, which again calls on forward-backward tracking with the `fbtrack(...)` function. This function computes the Lucas-Kanade optical flow from the previous frame to the current frame, and then the other way around. A proposed candidate for the new bounding box is returned by the tracker.

Following, the Detection module is launched with the function `detect(img=grey)`, where foreground is calculated with the function `foregroundDetector::nextIteration(img)` (from background subtraction), before the patches(windows) is sent through the Variance filter, Ensemble Classifier, and Nearest Neighbor Classifier as explained in Section 3.2. The detector returns another candidate for the new bounding box, and both candidates is then sent to the `fuseHypotheses()` function. In this function, the candidates are evaluated and the bounding box with the highest confidence is chosen.

If the trackers candidate is chosen, it is considered for validation. If the tracked bounding box is valid, it is brought to the Learning module represented by the function `learn()`, where the bounding box' underlying patch is stored as a positive patch together with overlapping patches. Negative patches is computed in this function as well. The positive patches is sent to the Ensemble Classifier while the negative patches is sent to the Nearest Neighbor Classifier, for further use in the Detection module.

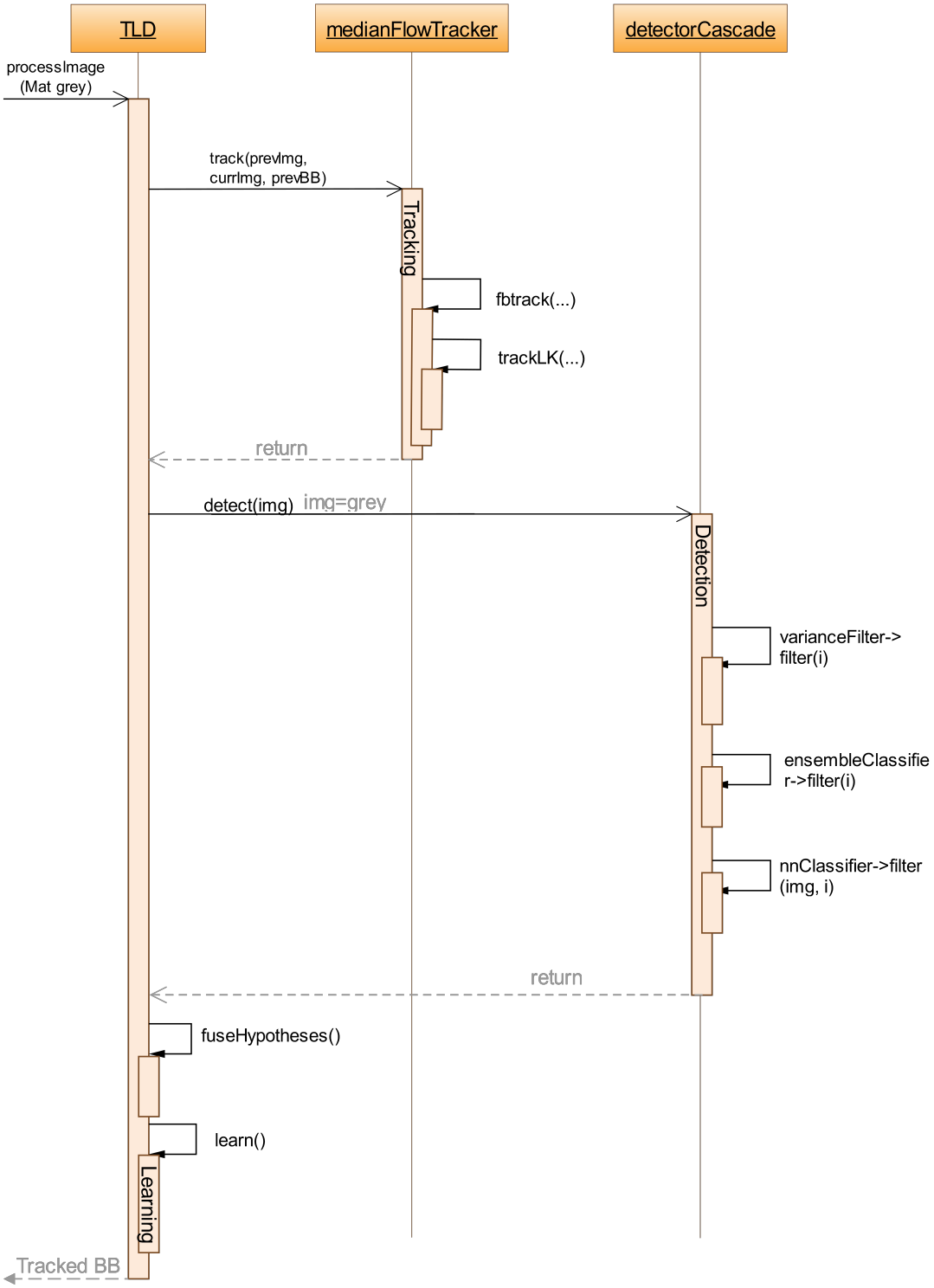

Figure 7.2: Implementation: OpenTLD - Sequence diagram of the OpenTLD implementation.

Part III

System Evaluation

8

Performance Analysis

As mentioned in the project description, the critical factors considered for the application's performance is memory usage and processing time, due to the limited computational power in mobile devices. The quality of the tracking, such as precision and recall measures, is not evaluated in this thesis, as these measures is thoroughly evaluated in [14] and [37]. In this chapter, there will be analysis of memory usage and run-time, and how they depend on a selection of relevant variables. The selected variables is; number of features from the Tracking module, size of the produced bounding box, and number of patches that were registered as valid and stored in the Learning module. Before testing, a few initial hypotheses were derived about dependencies between the variables and the performance.

1. **Amount of tracked feature points;** would affect the Tracker module's run-time, due to more points to compare in the Lucas-Kanade algorithm. Also the rendering could take longer.

2. **Size of bounding box;** would affect the Detector module's run-time, due to larger patches containing more pixels being sent for matching in the Cascaded Classifier(see Section 3.2). Could also affect the memory usage.

3. **Number of learned patches;** would affect memory and run-time performance. Run-time of the Learning module should be affected, due to processing needed to learn new patches(extraction of patch, computing overlap, classifying patch). Run-time of the Detector module should increase as there will be more patches

to match. The memory usage should also be affected due to the growing amount of stored patches.

Another factor which obviously would affect the performance, is the size of the processed image frames. The default size is 800 x 480 on the used device, and in Figure 8.1, the quality of the different scales used can be observed.

Figure 8.1: Scales - Subimages from frames with the different scales. Scale 1: 800 x 480, Scale 2: 400 x 240, Scale 4: 200 x 120. The lower-resolution scales is obtained by downsampling the original image with FastCV's fcvScaleDownXX functions(XX equals scale specifiaction). Visually the images becomes blurred.

In the following sections, the test results will be presented. Section 8.1 contains analysis of the memory usage, while the processing time will be analysed in Section 8.2. Each of the analyses is supported by measures of the mentioned variables, for evaluation of the listed hypotheses in Chapter 9.

8.1 Memory Usage

To measure memory usage, Android has several useful tools. In this project the classes ActivityManager and Debug is used. The ActivityManager class contains overall functionality to interact with the running activities, and for obtaining memory information, the underclass MemoryInfo can be used. The functionality in this underclass is limited, but is used in the following analysis to measure the available memory on the targeted device. The class Debug, also contains a subclass called MemoryInfo(not to be confused with the just mentioned subclass of ActivityManager). This MemoryInfo subclass has more detailed memory measures, but is limited to the current application. To measure the memory used by this application, there is several to choose from, however PSS and Private Dirty seems sufficient for this project. PSS, or proportional set size, is a relative measure of the memory usage compared to the amount of other processes running.

Private Dirty is the memory inside the current process, which can not be paged to disc. [38]. These two measures can be found together with the available memory in Figure 8.2.

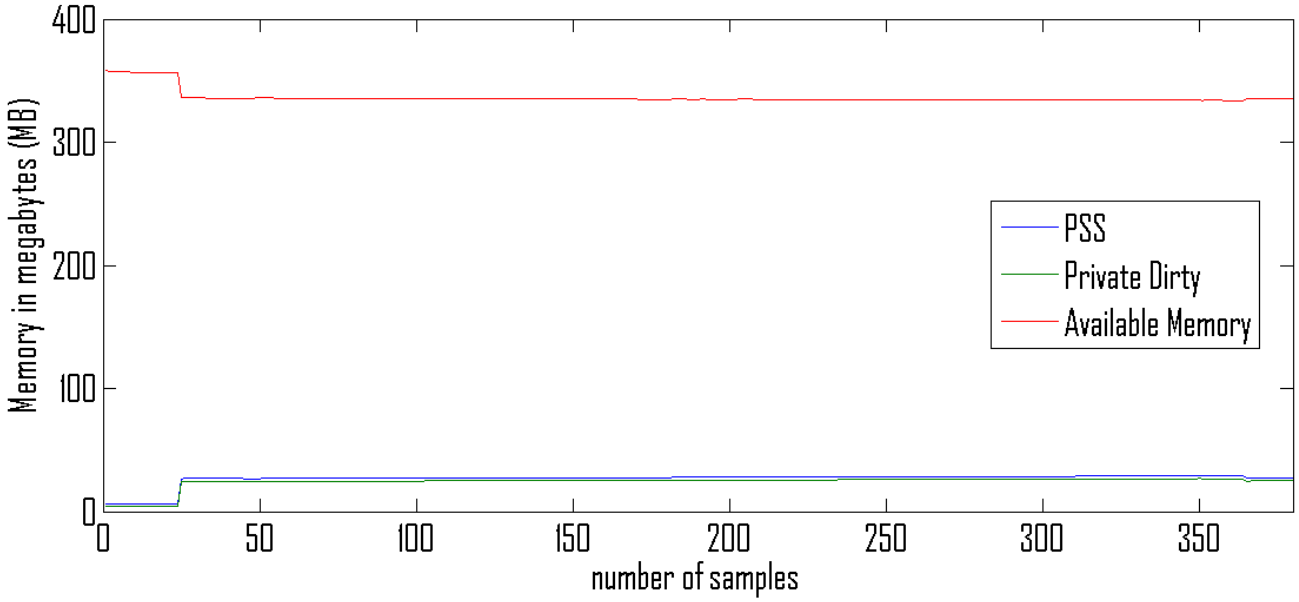

Figure 8.2: OpenTLD: Memory Analysis (Scale 4) - The first phase is before initialisation. The most drastic change is when initialisation occurs, where memory is allocated for the TLD functions. Timespan = 87,8 s.

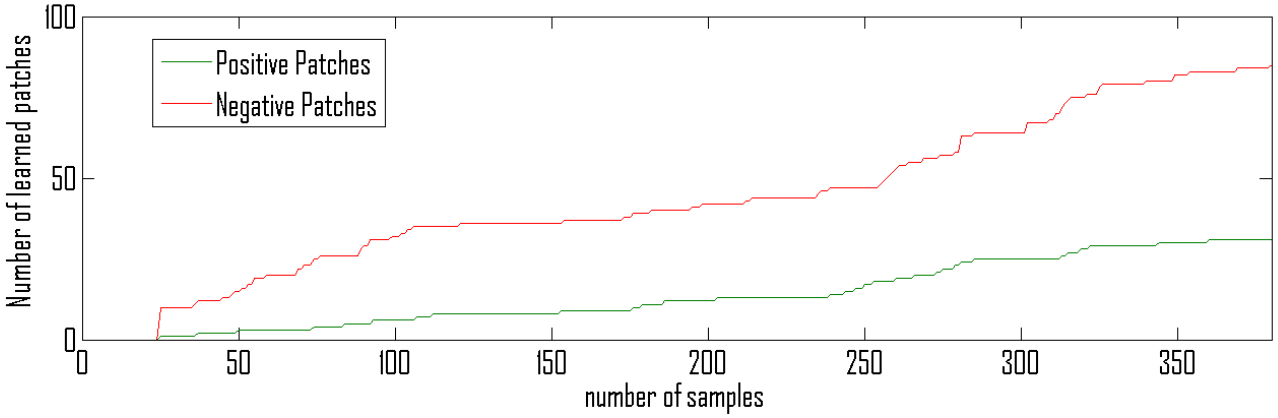

Figure 8.3: Learned patches (Scale 4) - Timespan = 87,8 s.

The extraction of memory data also affected the processing time significantly. Be-

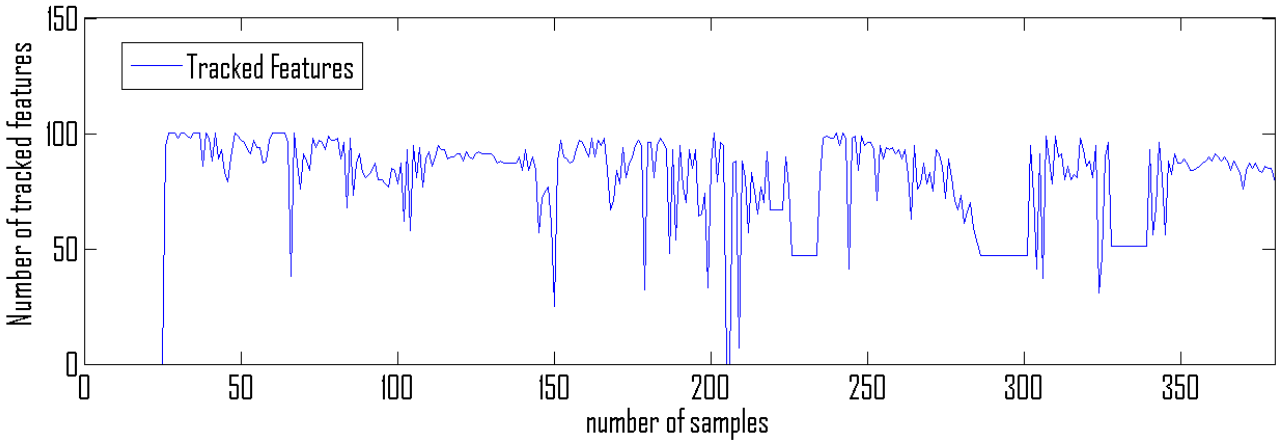

Figure 8.4: Tracked features (Scale 4) - Timespan = 87,8 s.

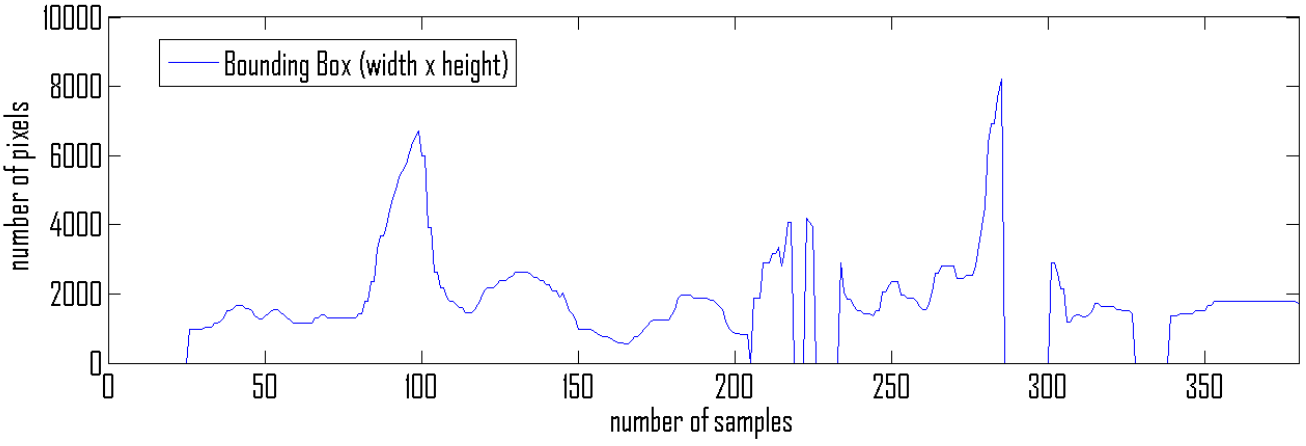

Figure 8.5: Bounding box size (Scale 4) - Timespan = 87,8 s.

low, in Figure 8.6 the run time is parted up in the same categories as in the evaluation of the initial systems' in Chapter 5. In addition the time processed during run of memory functions, is labeled as 'Memory Evaluation'. The time intervals is measured with the functions `nanoTime()` in Java and `getTimeMicroSeconds()` in C++ as before.

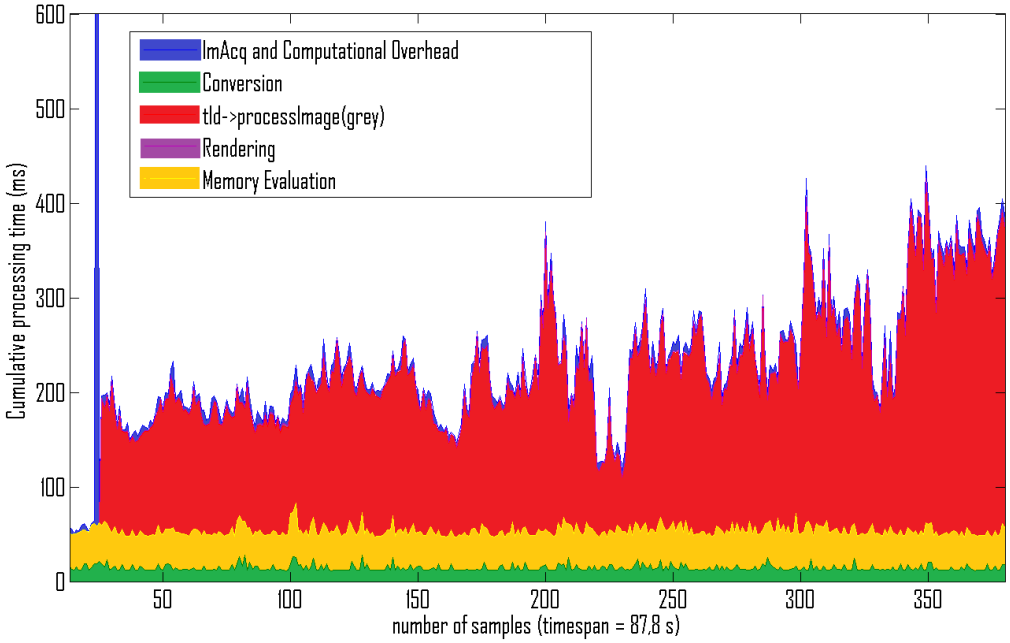

Figure 8.6: OpenTLD (Scale 4) - Processing time analysis.

8.2 Processing Time

The following analysis of the processing time is slightly different than the previously presented. The categories presented as Image Acquisition, Conversion, Rendering and Computational Overhead, which represents the functionality based on the FastCV sample, is now moved into one label. The focus for the analysis is the content of the processImage() function, which represent the OpenTLD functionality, grouped into the main modules Tracking, Learning and Detection. The first part of the results is from the system runned on scale 2 image frames, while in the second part, the system processed scale 4 images.

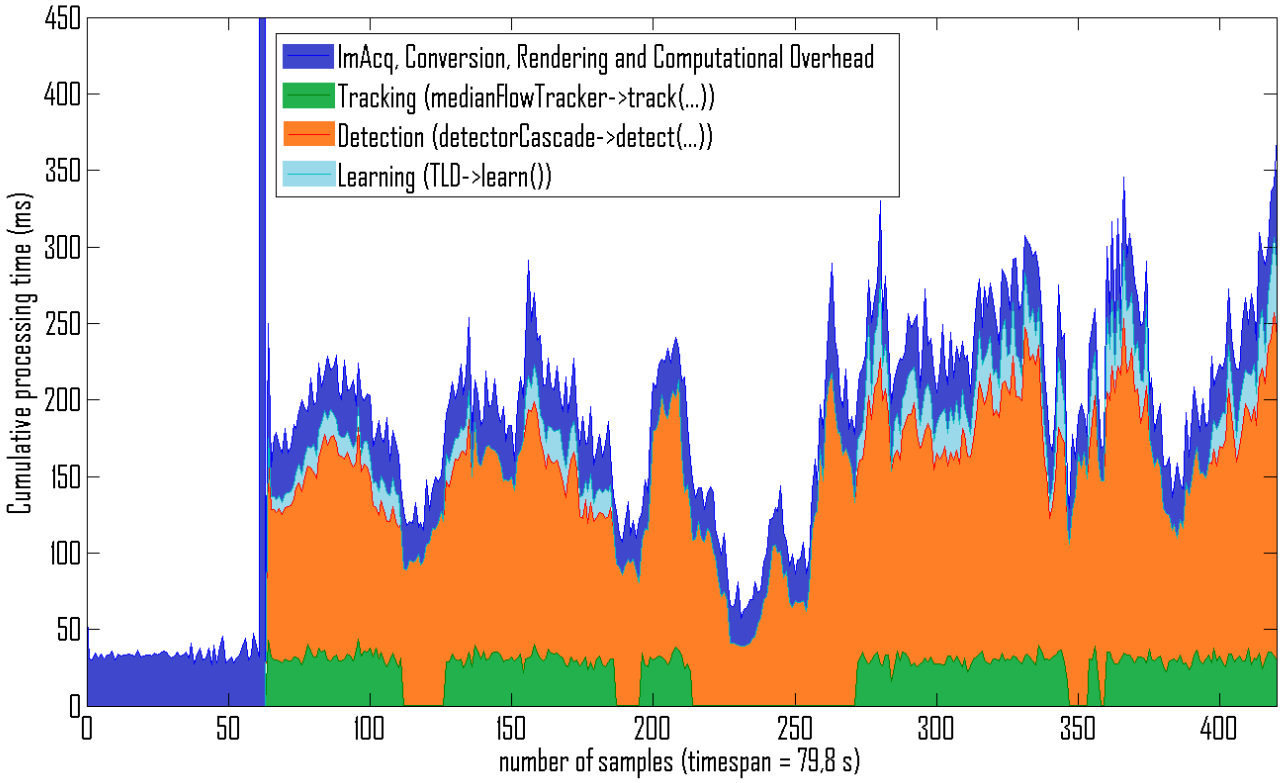

Figure 8.7: OpenTLD Processing Time (Scale 2) - The first phase until the first peak, is runtime before bounding box is selected. It's worth to notice that this phase usually will stabilize around 33 ms, as the camera preview's frame rate is 30 fps (this also means that the processing in the FastCV framework finishes before the new frame arrives). The first peak is the initialisation after the bounding box is selected(the peak usually ends at 2000-3000 ms). The periods when the tracker is not running, is when the object is lost(occluded, out of screen or just not detected by other reasons).

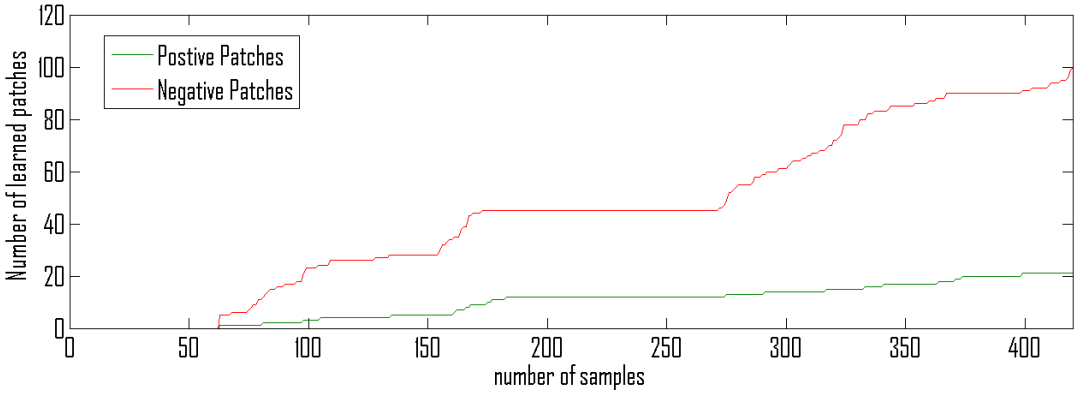

Figure 8.8: Learned Patches (Scale 2) - Timespan = 79,8 s.

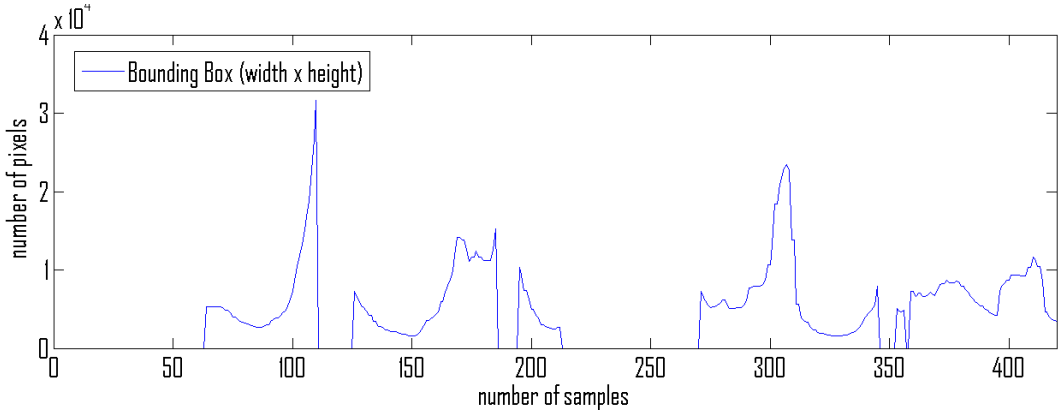

Figure 8.9: Bounding box size (Scale 2) - Timespan = 79,8 s.

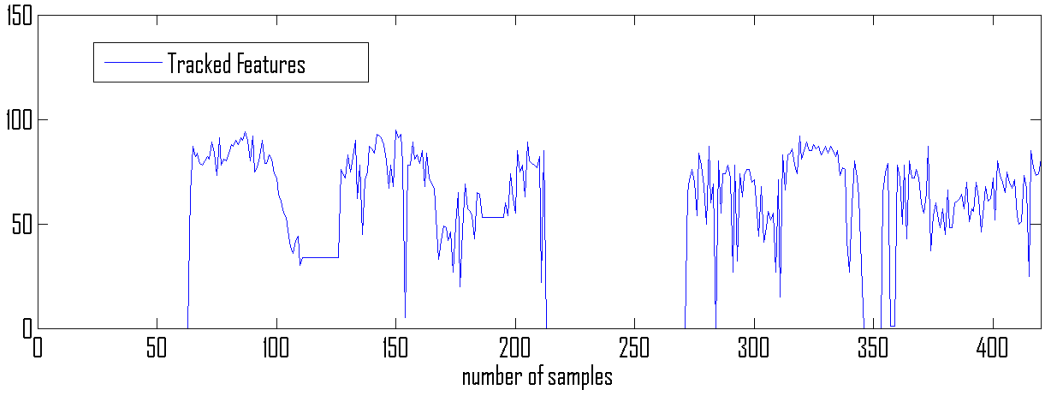

Figure 8.10: Tracked Features (Scale 2) - Timespan = 79,8 s.

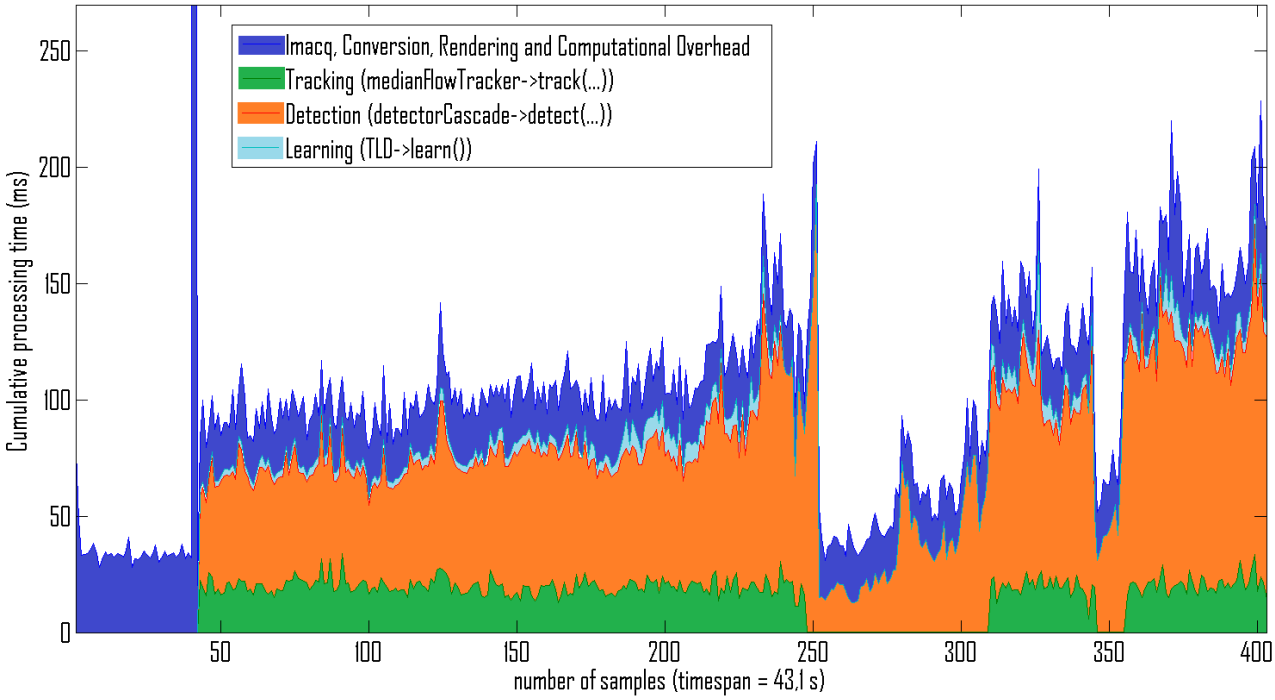

Figure 8.11: OpenTLD Processing Time (Scale 4) - The first phase until the first peak, is runtime before bounding box is selected. It's worth to notice that this phase usually will stabilize around 33 ms, as the camera preview's frame rate is 30 fps (this also means that the processing in the FastCV framework finishes before the new frame arrives). The first peak is the initialisation after the bounding box is selected(the peak usually ends at 2000-3000 ms). The periods when the tracker is not running, is when the object is lost(occluded, out of screen or just not detected by other reasons). (Copied from Figure 8.7 for convenience)

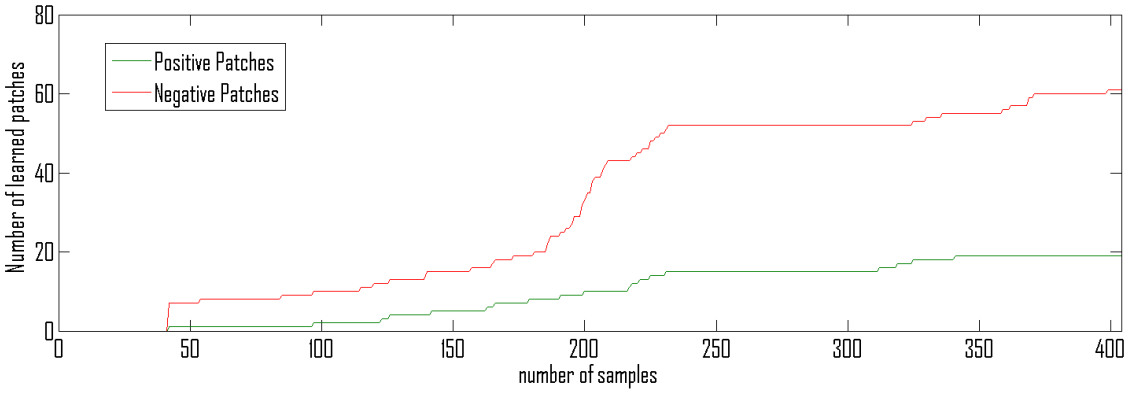

Figure 8.12: Learned Patches (Scale 4) - Timespan = 43,1 s.

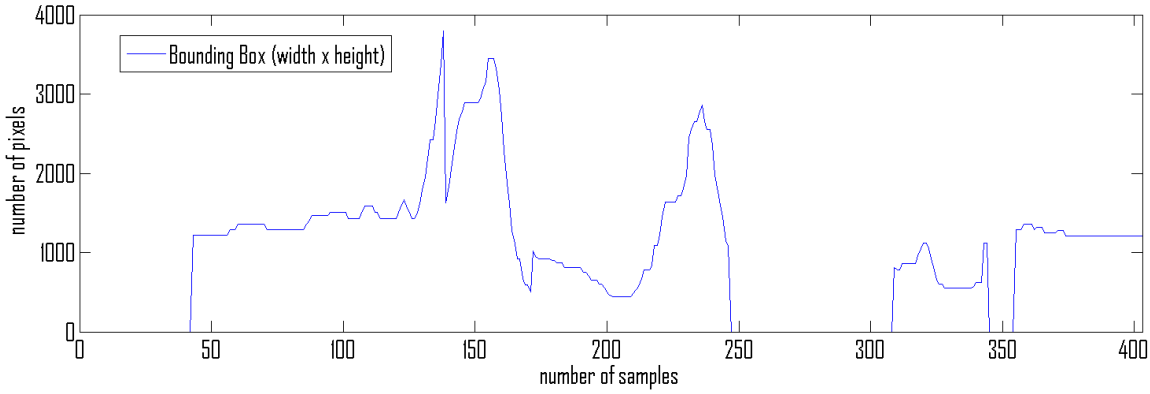

Figure 8.13: Bounding box size (Scale 4) - Timespan = 43,1 s.

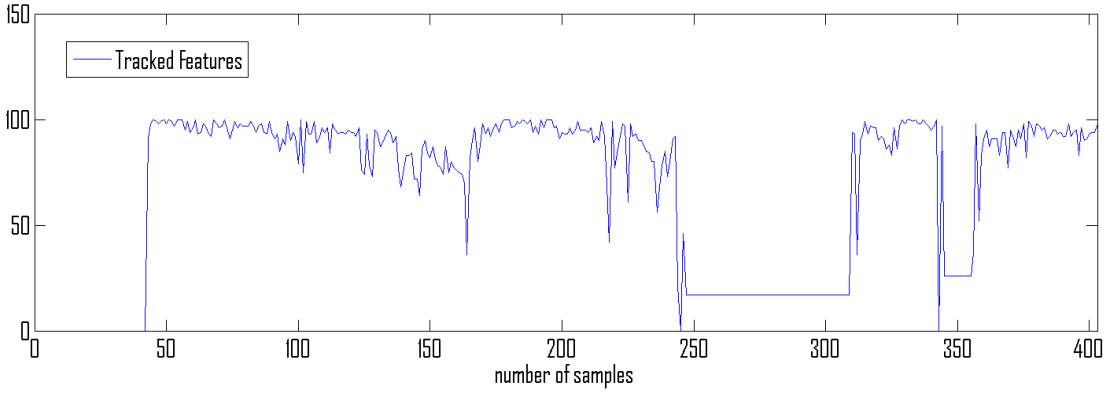

Figure 8.14: Tracked Features (Scale 4) - Timespan = 43,1 s.

9

Discussion

This chapter will discuss the performance analysis from Chapter 8 and evaluate the initial hypotheses about the variable dependencies. Firstly, there will be some comments about the memory usage, before processing time will be discussed. The hypotheses will be evaluated for the relevant analysis, before other remarks will end this chapter.

The memory usage was thought to be a potential issue, because of the many subimages, called patches, that should be stored in the Learning phase. From Kalal et al. , it was observed that the amount of stored patches stabilised on several hundred, and it should not be a significant risk for memory shortage on regular computers. However, with the more limited amount of memory on mobile devices, it made sense to investigate the memory usage for TLD on the Android platform. From Figure 8.2 it can be seen, when the object tracker begins(after about 25 samples), there is an increase in the memory usage of the active process(PSS and Priv. Dirty), and a correspondent reduction in the available memory. After this point, there is a slight decrease in the available memory, but does not come close to memory shortage. To adress the hypotheses; there is no visible conection between the size of the bounding box and the memory analysis, however there is possibly a significant dependency between number of patches learned and memory usage. This test of memory usage was performed with the scale 4 images, but it is performed a similar test with the scale 2 images (in Appendix 12.2), and no significant difference was observed regarding memory usage.

The analysis of processing time can be divided in two parts; testing with scale 2 images and testing with scale 4 images. The cumulative processing time with scale 2 images and scale 4 images, can be seen in Figure 8.7 and Figure 8.11 respectively.

The most evident change in the graphs is noticed when the Tracking module stops (object is defined as lost by the tracker). This event drastically reduces the processing time due to the following reasons; 1. the Tracking module's processing is skipped, 2. there is no learning without tracking and 3. less patches is going through the Nearest Neighbor Classifier, because few patches, if any, lets through the Ensemble Classifier in the Detection module. Following is a list, to address the initial hypotheses regarding the variables:

1. **Amount of tracked feature points;** does not affect the Tracker module's run-time to a significant degree. The same is true for rendering(see Figure 8.6).

2. **Size of bounding box;** does not affect the Detector module's run-time to a significant degree, at least not in a consistent manner. The influence on memory usage, is not noticeable either like already mentioned.

3. **Number of learned patches;** is affecting both memory and run-time. The memory usage seems to be slightly influenced by this, but not to a critical level. On the other hand, run-time is dependent on the amount of learned patches to a larger extent. This can be seen for both scale 2 and scale 4, where the Detector module's processing time tends to grow in both cases, as the amount of patches increases. The Learning module is also affected by the increase of patches (more precisely, the gradient of the graph).

From testing with image scale 2 compared to image scale 4, it is experienced that while image scale 4 is more than twice as fast, it also lets the bounding box drift slightly more(see Figure 6.1 from the user perspective example, where there is a drift between the initial and the tracked bounding box.). For scale 2, the detector will process images with more details, which will make the detection more accurate, due to more unique results from the Ensemble Classifier. On the other hand, the tracker will give about half as many positive patches to the learning module, thus it will take longer to make the classifiers robust.

Another remark experienced through testing, is that video cameras have a mechanism to turn down its image acquisition speed during low light conditions. This may affect test results from day to day, depending on the weather and time of the day. The

test results presented in the performance analysis, however, is taken on the same day and shouldn't be affected too much from this.

10

Conclusion

During this project a long-term tracker, based on the Tracking-Learning-Detection(TLD) algorithm, has been ported to the Android platform. The resulting application is made out of a C++ version of TLD, OpenTLD, which is integrated into a framework based on the FastCV sample application. The performance, with regard to memory and processing time, is analysed in Chapter 8. From this analysis, the system runs with a frame-rate about 10 fps (with scale 4 images), and OpenTLD is observed to dominate the processing time, which is desired. However, the processing time is increasing after a period of time, especially the Detector module, due to a growing amount of patches in the learning module. Suggested solution to this is mentioned in Further Work. Nonetheless, the object tracker application works as intended on the Android platform and could possibly be used for Augmented Reality (See videos on the attached CD, to get another impression of how it performs).

The motivation for making this object tracker for Android, was to utilize it for the Augmented Reality purpose. However, other technological concepts which is not discussed, could also exploit the OpenTLD functionality. This could be system's for Human-Computer Interaction (HCI), Video Stabilization or as visual input for robotic control.

11

Further work

For further work, the processing time could be reduced even more, possibly by making a longer interval between each run with the detector module. The system response is clearly becoming slower, as the amount of patches increases. As mentioned in [14], when the amount reaches a threshold, the system starts to randomly remove patches. This threshold could possibly be set lower for the Android version to limit the speed reduction, but this has not been investigated yet.

G. Nebehay introduced functionality for loading a previously learned object model, but this is not yet a feature for the Android version. This should be looked into though, as it would be useful in many situations, such as in an Augmented Reality application, where the objects usually is predetermined.

Like discussed by Wagner[31], it is possible to trade memory for speed and robustness in the Fern Classification (Ensemble classifier). In this project, however, the memory usage was not the critical factor, and this could mean that it would make sense to trade speed and robustness for memory instead.

It would also be interesting to see a analysis of the tracking quality (ie. precision/recall measures), when doing this tradeoff or when reducing the image resolution.

References

[1] ISMAR09 mobile committee. **History of Mobile Augmented Reality**. *https://www.icg.tugraz.at/ daniel/HistoryOfMobileAR/*, 2012. 1

[2] N. Bilton New York Times. **Google Begins Testing Its Augmented-Reality Glasses**. *http://bits.blogs.nytimes.com/2012/04/04/google-begins-testing-its-augmented-reality-glasses/*, 2012. 1

[3] R. T. Azuma. **A Survey of Augmented Reality**. *Teleoperators and Virtual Environments*, 1997. 7

[4] Asad-Uj-Jaman. **Sensors in Smartphones**. *http://mobiledeviceinsight.com/2011/12/sensors-in-smartphones/*, 2011. 7

[5] wikitude.com. **Wikitude: Statue of Liberty**. *http://www.wikitude.com/tour/wikitude-world-browser/statue-of-liberty*. 9

[6] Shotzoom Software. **Golfscape Brings Augmented Reality Golf Rangefinder to iPhone 4**. *http://www.bizjournals.com/prnewswire/press_releases/2010/07/14/LA33990*, 2010. 9

[7] Qualcomm. **QDevNet - Vuforia: System Overview**. *https://ar.qualcomm.at/qdevnet/developer_guide/System%20Overview*. 11

[8] S. Choudhury. **Snapdragon Insider Blog: Introducing FastCV, Computer Vision Technology Tuned for Mobile**. *http://www.qualcomm.com/snapdragon/media/blog/2011/10/25/introducing-fastcv-computer-vision-technology-tuned-mobile*, 2011. 12

[9] Qualcomm Development Team. **Qualcomm FastCV Library v1.0: Modules**. *https://developer.qualcomm.com/docs/fastcv/api/modules.html*, 2012. 13

[10] OpenCV Development Team. **OpenCV documentation**. *http://docs.opencv.org/*, 2012. 13

[11] ARToolWorks. **www.artoolworks.com: About Us**. *http://www.artoolworks.com/corporate/about-us/*. 14

[12] G. Klein and D. Murray. **Parallel Tracking and Mapping for Small AR Workspaces**. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007. 14

[13] H. Durrant-Whyte and T. Bailey. **Simultaneous localization and mapping: part I**. *Robotics Automation Magazine, IEEE*, **13**(2):99 –110, june 2006. 14

[14] Z. Kalal, K. Mikolajczyk, and J. Matas. **Tracking-Learning-Detection (Accepted, but not published yet**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PP**(99):1, July 2012. 15, 16, 21, 22, 25, 26, 34, 55, 71

[15] Z. Kalal, J. Matas, and K. Mikolajczyk. **P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints**. *Conference on Computer Vision and Pattern Recognition*, 2010. 15, 17

[16] J. Matas. **Homepage: Jiri Matas**. In *http://cmp.felk.cvut.cz/ matas/*. 15

[17] B. D. Lucas and T. Kanade. **An Iterative Image Registration Technique with an Application to Stereo Vision**. *International Joint Conferences on Artificial Intelligence*, pages 674–679, 1981. 17

[18] S. Baker and I. Matthews. **Lucas-Kanade 20 Years On: A Unifying Framework**. *International Journal of Computer Vision*, **56**:221–255, 2004. 17, 19, 20

[19] H. Grabner and H. Bischof. **On-line Boosting and Vision**. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, **1**, pages 260 – 267, june 2006. 26

[20] H. Grabner, C. Leistner, and H. Bischof. **Semi-supervised On-Line Boosting for Robust Tracking**. In *Computer Vision ECCV 2008*, **5302**, pages 234–247. 2008. 26

[21] S. Stalder, H. Grabner, and L. van Gool. **Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition**. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1409 –1416, 27 2009-oct. 4 2009. 26

[22] B. Babenko, Ming-Hsuan Yang, and S. Belongie. **Visual tracking with online Multiple Instance Learning**. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983 –990, june 2009. 26

[23] Q. Yu, T. Dinh, and G. Medioni. **Online Tracking and Reacquisition Using Co-trained Generative and Discriminative Trackers**. In *Computer Vision ECCV 2008*, **5303**, pages 678–691. 2008. 26

[24] C. Pettey. **Gartner Newsroom: Sales of Mobile Devices**. *http://www.gartner.com/it/page.jsp?id=1848514*, 2011. 27

[25] developer.android.com. **The Developer's Guide: Installing the SDK**. *http://developer.android.com/sdk/installing.html*. 28

[26] developer.android.com. **The Developer's Guide: Download the Android NDK**. *http://developer.android.com/sdk/ndk/index.html*. 28

[27] E. Gravdal. **Object Recognition and Outlier Detection for Mobile Phones**. *NTNU - TTK4550 - Engineering Cybernetics, Specialization Project*, December 2011. 34

[28] E. Rosten and T. Drummond. **Fusing points and lines for high performance tracking**. **2**:1508 –1515 Vol. 2, oct. 2005. 34

[29] E. Rosten and T. Drummond. **Machine Learning for High-Speed Corner Detection**. **3951**:430–443, 2006. 34

[30] P. Viola and M. Jones. **Rapid object detection using a boosted cascade of simple features**. **1**:I–511 – I–518 vol.1, 2001. 34

[31] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. **Real-Time Detection and Tracking for Augmented Reality on Mobile Phones**. *Visualization and Computer Graphics, IEEE Transactions on*, **16**(3):355 –368, may-june 2010. 34, 71

[32] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. **Pose tracking from natural features on mobile phones**. pages 125–134, 2008. 34

[33] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. **Fast Keypoint Recognition Using Random Ferns**. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **32**(3):448 –461, march 2010. 34

[34] M. Ozuysal, P. Fua, and V. Lepetit. **Fast Keypoint Recognition in Ten Lines of Code**. pages 1 –8, june 2007. 34

[35] I. Khvedchenia. **Comparison of the OpenCVs feature detection algorithms**. *http://computer-vision-talks.com/2011/01/comparison-of-the-opencvs-feature-detection-algorithms-2/*. 34

[36] D. G. Lowe. **Distinctive Image Features from Scale-Invariant Keypoints**. *International Journal of Computer Vision*, **60**:91–110, 2004. 34

[37] G. Nebehay. **Robust Object Tracking Based on Tracking-Learning-Detection**. *MSc Thesis (draft version)*, 2012. 35, 46, 55

[38] hackbod. **StackOverflow: Android Memory Usage**. *http://stackoverflow.com/questions/2298208/how-to-discover-memory-usage-of-my-application-in-android*, 2010. 57

11

12

Appendix

12.1 Device Used

Figure 12.1: Samsung Galaxy S II - Info is taken from `http://en.wikipedia.org/wiki/Samsung_Galaxy_S_II`

- **Model:** GT-I9100
- **Android version:** 4.0.3 (Ice Cream Sandwich)
- **SoC:** Samsung Exynos 4 Dual 45nm
- **CPU:** 1.2 GHz dual-core ARM Cortex-A9
- **GPU:** ARM Mali-400 MP
- **Memory:** 1 GB RAM
- **Display:** 4.3 in (110 mm) AMOLED

12.2 Obsolete Test Results

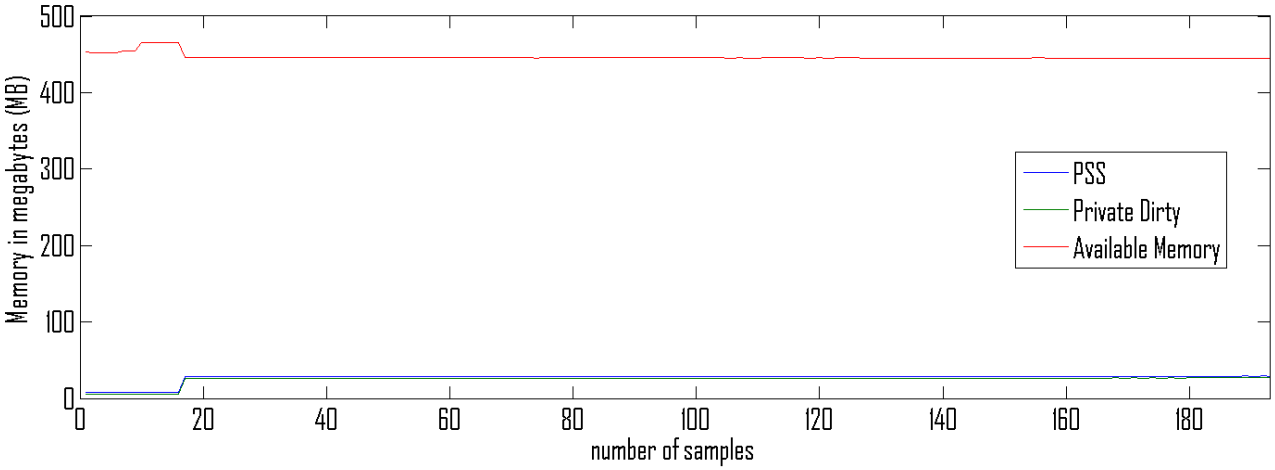

Figure 12.2: OpenTLD: Memory Analysis (Scale 2) - The first phase is before initialisation. The most drastic change is when initialisation occurs, where memory is allocated for the TLD functions. Timespan = 57,1 s.

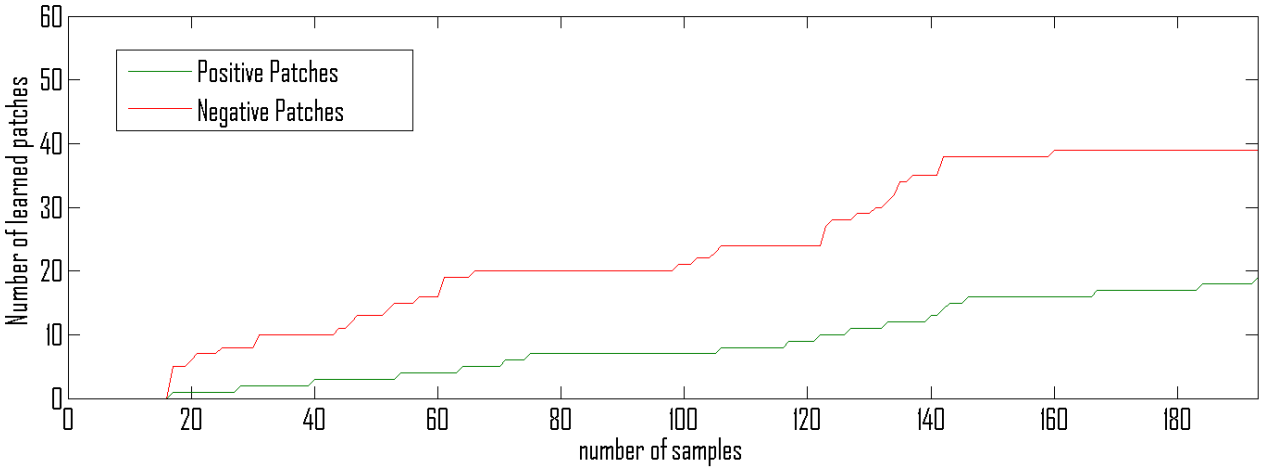

Figure 12.3: Learned patches (Scale 2) - Timespan = 57,1 s.

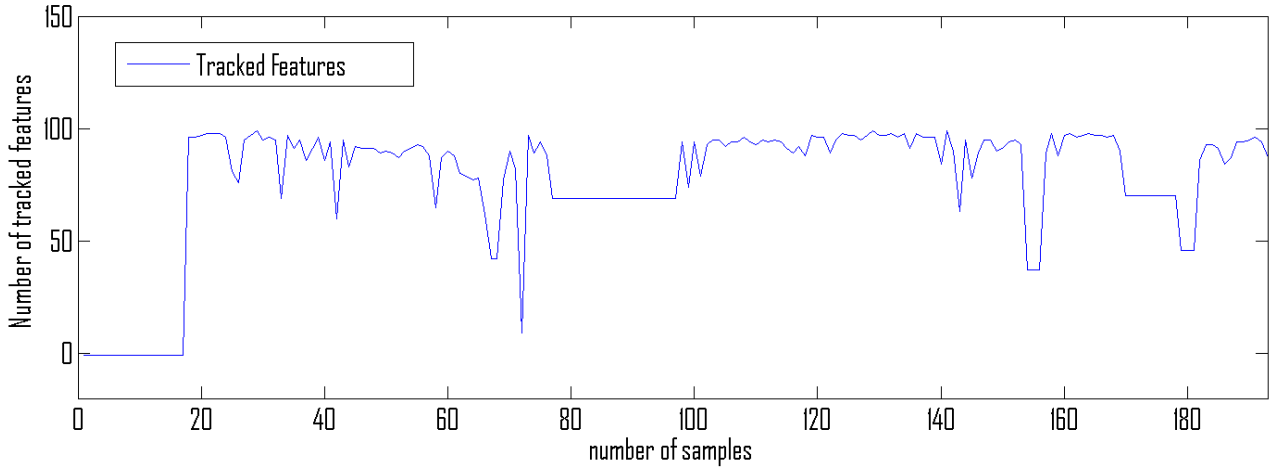

Figure 12.4: Tracked features (Scale 2) - Timespan = 57,1 s.

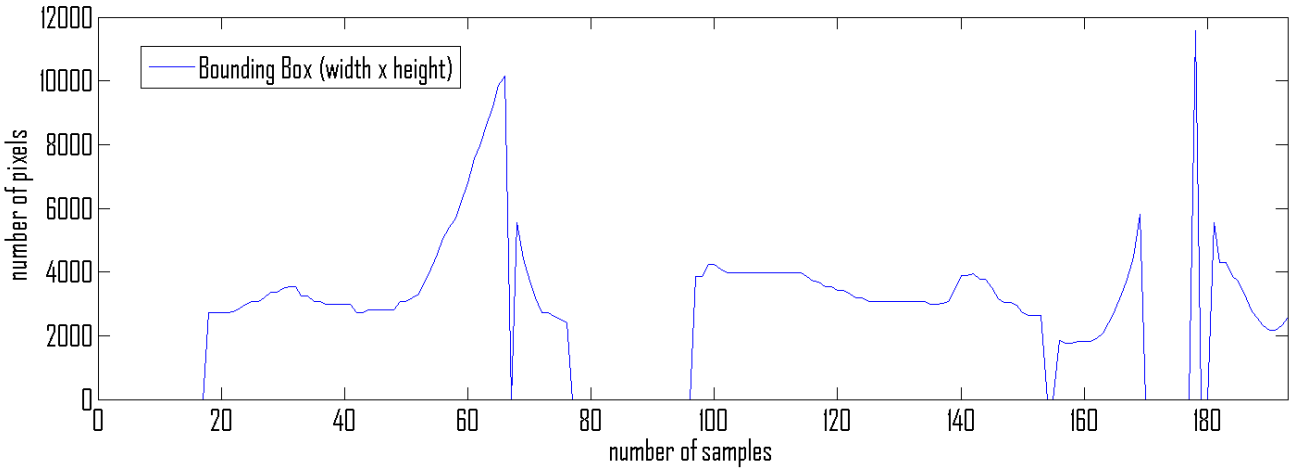

Figure 12.5: Bounding box size (Scale 2) - Timespan = 57,1 s.

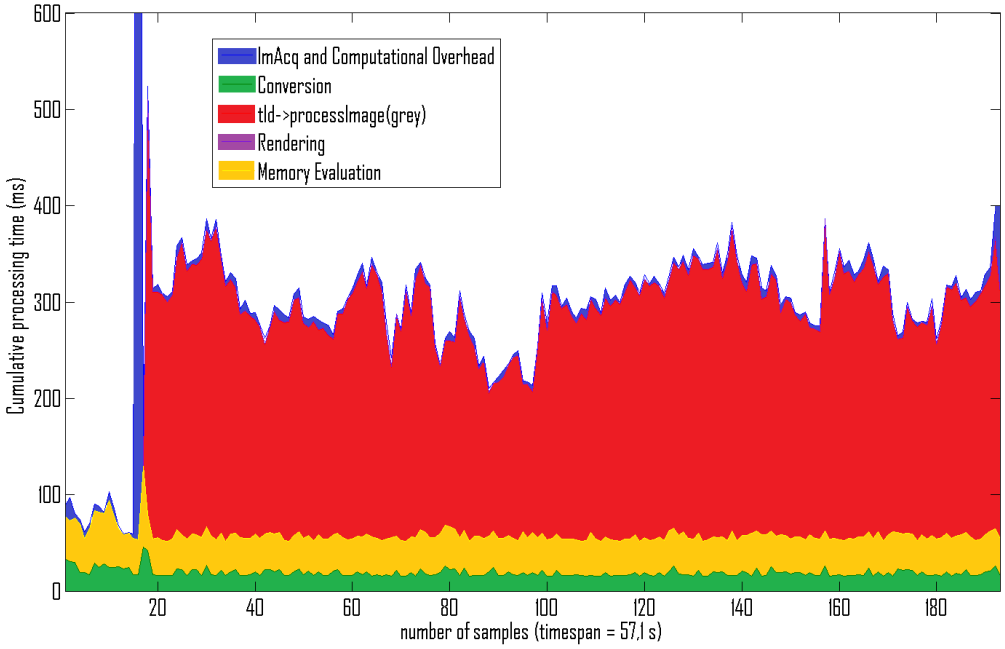

Figure 12.6: OpenTLD (Scale 2) - Processing time analysis.

12.3 Vuforia AR marker

To demonstrate the possibilities with the Vuforia SDK, this Appendix section contains an AR maker, which is used by some AR applications. The House Model application can be downloaded from `https://play.google.com/store/apps/details?id=org.monosock.shadowdemo&hl=en`, and the billiard application '3D Pool game' can be downloaded from `https://play.google.com/store/apps/details?id=com.rrrstudio.billiards&hl=en`

Figure 12.7: Vuforia AR Marker -