



M 2015

MOBILE OBJECT TRACKER

JOSÉ GUILHERME AZEVEDO MARQUES PEREIRA

DISSERTAÇÃO DE MESTRADO APRESENTADA

À FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO EM

MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Mobile Object Tracking

José Guilherme Azevedo Marques Pereira

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Filipe Pinto de Almeida Teixeira (PhD)

Second Supervisor: João Nuno Castro Gonçalves (MSc)

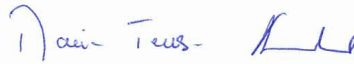
June 29, 2015

A Dissertação intitulada

“Mobile Object Tracker”

foi aprovada em provas realizadas em 14-07-2015

o júri



Presidente Professora Doutora Maria Teresa Magalhães da Silva Pinto de Andrade
Professora Auxiliar do Departamento de Engenharia Eletrotécnica e de
Computadores da Faculdade de Engenharia da Universidade do Porto



Professor Doutor António José Ribeiro Neves
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática
da Universidade de Aveiro



Professor Doutor Luís Filipe Pinto de Almeida Teixeira
Professor Auxiliar do Departamento de Engenharia Informática da Faculdade de
Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.



Autor - José Guilherme Azevedo Marques Pereira

Faculdade de Engenharia da Universidade do Porto

Resumo

Actualmente, a maioria dos sistemas de vigilância existentes no mercado utilizam, para motivos de *tracking* de objetos, técnicas de subtracção de *background*. A utilização dessas técnicas simplesmente permitem que estes sistemas sejam capazes de ser utilizados em ambientes estáticos, levando a uma diminuição do seu uso numa vasta possibilidade de cenários, cujo ambiente é dinâmico. De maneira a combater esse fenómeno, é necessário desenvolver um sistema portátil e que seja capaz de seguir objectos em ambiente dinâmicos.

Nos últimos anos, existem forças especiais de segurança que recorrem a sistemas de gravação de vídeo, de forma a gravar os acontecimentos ocorridos, e para, posteriormente, poderem ser analisados por equipas especiais. Nesse contexto, a Fraunhofer AICOS Portugal desenvolveu um projeto, denominado de SAFETY, que consiste na incorporação de um smartphone num colete desenhado para auxiliar as forças de segurança. O objectivo desta dissertação, que se encontra incorporado no projecto SAFETY, tem como intenção a criação de uma aplicação Android capaz de detectar e seguir pessoas na gravação de vídeo obtida pelo smartphone. O projecto desenvolvido adicionará uma funcionalidade extra ao projecto SAFETY e ajudará o dia-a-dia das forças de segurança, fornecendo gravações mais detalhadas das situações encontradas.

A aplicação consiste num sistema de *tracking* em tempo real e está dividido em duas etapas principais: o detector automático de pessoas e o algoritmo de *tracking*. O detector de pessoas implementado baseia-se na utilização do algoritmo *Histogram of Oriented Gradients* (HOG) com a ajuda de um classificador, previamente treinado com o INRIA dataset, para detectar positivamente pessoas, enquanto que o algoritmo de *tracking* recorre a dois algoritmos: *Tracking Learning Detection* (TLD) ou *Consensus-based Matching and Tracking of Keypoints* (CMT).

De maneira à aplicação ser testada em termos da sua precisão, é necessário a utilização de um dataset desenvolvido para esse efeito. A análise dos resultados foi efectuada qualitativamente e quantitativamente com a ajuda do dataset utilizado e sequências obtidas através da aplicação implementada. Os resultados obtidos foram positivos e promissores, contudo algumas funcionalidades deste projeto podem vir a ser melhorados num futuro trabalho.

Abstract

Nowadays, most of the surveillance systems existent in the market rely in background subtraction techniques for tracking purposes. Therefore, the system just has the ability to work in environments with static background, diminishing the array of scenarios where this type of applications can be used. To counteract that phenomenon, there is the need to develop a portable system which is capable of tracking objects in dynamic scenarios.

In recent years, exist special security forces that have employed in their equipment visual recording systems to posteriorly analyze the obtained data. In that sense, Fraunhofer AICOS Portugal developed a project called SAFETY which incorporates a vest with an integrated smart-phone to aid security forces in their daily tasks. The objective of this dissertation has to do with the SAFETY project and aims to develop an Android application capable of detecting and tracking people along the video frames captured. This project will improve the SAFETY project with an extra function which will help the security forces analyze teams with video recordings with more information, facilitating their job.

The application consists of a real-time tracking system and is divided in two important stages: automatic people detector and tracking algorithm. The people detector is implemented using the Histogram of Oriented Gradients (HOG) algorithm that relies in a Support Vector Machine (SVM) classifier trained by the INRIA dataset to more accurately detect people, while the tracking algorithm relies in two different algorithms: Tracking Learning Detection (TLD) or Consensus-based Matching and Tracking of Keypoints (CMT).

In order to test this application properly, a dataset already developed by other parties was used to determine the accuracy of this application. The analysis of the results was made qualitatively and quantitatively with the help of a dataset and some sequences recorded using the Android application. The obtained application results were positive and promising, yet some features of this project can still be improved in a future work.

Acknowledgements

I would like to thank the institutions of Fraunhofer AICOS Portugal and Faculty of Engineering of University of Porto for providing a suitable environment for the realization of this dissertation and the academic background obtained over the last few years.

Prof. Luís Teixeira and Eng. João Gonçalves were key elements in the supervision and guidance during the ongoing project. Therefore, I would like to extend my gratitude for their availability and cooperation.

Last but not least, I want to express my thankfulness to all my friends and family who supported me through this stage of my academic life.

José Guilherme Pereira

“I’m Feeling Lucky”

Google

Contents

1	Introduction	1
1.1	Context	1
1.2	Objectives	2
1.3	Contributions	3
1.4	Structure	3
2	State of the Art	5
2.1	Computer Vision in Object Tracking	5
2.1.1	Kernel-based Tracking	5
2.1.1.1	Mean Shift	6
2.1.1.2	Kanade-Lucas-Tomasi Feature Tracker	8
2.1.2	Point-tracking	8
2.1.2.1	Kalman Filter	9
2.1.2.2	Particle Filter	10
2.1.3	Local Features	11
2.1.3.1	Scale Invariant Feature Transform	12
2.1.3.2	Histogram of Oriented Gradients	13
2.1.4	Summary	15
2.2	Computer Vision Software	15
2.3	Conclusions	16
3	Real-time Tracking System of Unknown Objects	19
3.1	Architecture	19
3.2	Software Platforms and Portability	21
3.2.1	UNIX based Software	21
3.2.2	Android OS	23
3.3	Object Tracking Software	25
3.3.1	Tracking-Learning-Detection	26
3.3.1.1	Tracking	27
3.3.1.2	Detection	28
3.3.1.3	Learning	33
3.3.2	CMT	34
3.3.2.1	Matching and Tracking of Keypoints	35
3.3.2.2	Voting	36
3.3.2.3	Consensus	37
3.3.3	Software Implementation	38
3.4	Discussion	39

4	Evaluation	41
4.1	Evaluation Method	41
4.2	Dataset	43
4.3	Results	44
4.3.1	Qualitative Results	44
4.3.1.1	Dataset Results	44
4.3.1.2	Android Application Results	47
4.3.2	Quantitative Results	53
4.4	Discussion	55
5	Conclusions and Future Work	57
5.1	Future Work	58
	References	59

List of Figures

2.1	Mean Shift analysis [10]	6
2.2	Mean Shift tracking results [11]	7
2.3	KLT tracking results [4]	8
2.4	Kalman filter model	9
2.5	SIR algorithm model [20]	11
2.6	Local features recognition procedure [5]	12
2.7	Overview of HOG feature extraction and object detection diagram [23]	14
3.1	System framework diagram	20
3.2	Software project flow	21
3.3	OpenCV modules	22
3.4	C/C++ project flowchart diagram	23
3.5	OpenCV Java API application structure	24
3.6	OpenCV application structure using NDK	25
3.7	TLD framework diagram [2]	26
3.8	OpenTLD architecture	27
3.9	Recursive tracking method [29]	27
3.10	Detection cascade approach [28]	28
3.11	Variance filter output [28]	30
3.12	Single fern feature calculation [28]	31
3.13	Distance classification of unknown patches [28]	32
3.14	High confidence sub-windows (yellow) and true detection (green) [28]	32
3.15	P/N Constraints [28]	34
3.16	CMT tracker principles [3]	35
4.1	Overlap measurement [28]	41
4.2	Overlap comparison cases [28]	42
4.3	TLD algorithm output (green region) over the BoBot video sequences of A, B, C, E, G, I and K (top to bottom) from frames 1, 100, 200 and 300 (left to right)	45
4.4	CMT algorithm output (blue region) over the BoBot video sequences of A, B, C, E, G, I and K (top to bottom) from frames 1, 100, 200 and 300 (left to right)	46
4.5	Sequence A1	47
4.6	Sequence A2	48
4.7	Sequence B1	48
4.8	Sequence B2	48
4.9	Sequence C1	49
4.10	Sequence C2	49
4.11	Sequence D1	49

4.12 Sequence D2	50
4.13 Sequence E	50
4.14 Sequence F	51
4.15 Sequence G	51
4.16 Sequence H	52
4.17 Sequence I	52

List of Tables

2.1	Computer Vision software performance benchmark (higher is better)	16
4.1	Description of BoBot Sequences	43
4.2	Image sequences obtained from MOTrack application	47
4.3	TLD performance with three different values of ω . The values exhibited are respectively the precision and the recall values.	53
4.4	CMT performance with three different values of ω . The values exhibited are respectively the precision and the recall values.	53
4.5	Algorithms processing speed in frames per second (fps) using Desktop application	54
4.6	Algorithms processing speed in frames per second (fps) using MOTrack application	54

Abbreviations

INN	Nearest Neighbour Classification
ADT	Android Development Tools
API	Application Programming Interface
BoBoT	Bonn Benchmark on Tracking
CAMSHIFT	Continuously Adaptive Mean Shift
CMT	Consensus-based Matching and Tracking of Keypoints
DoG	Difference of Gaussians
FPS	Frames Per Second
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
IDE	Integrated Development Environment
JNI	Java Native Interface
KLT	Kanade-Lucas-Tomasi
MATLAB	Matrix Laboratory
NDK	Native Development Kit
NCC	Normalized Correlation Coefficient
OpenCV	Open Source Computer Vision
OS	Operating System
ROI	Region of Interest
SIFT	Scale Invariant Feature Transform
SIR	Sequential Importance Resampling
SIS	Sequential Importance Sampling
SVM	Support Vector Machine

Chapter 1

Introduction

Nowadays, with the exponential development of new technologies, smartphones have gradually become computationally more powerful and complex equipments. Every year, new models of smartphones arrive into the market, incorporating new and improved features providing new smartphone functions over the years. Among these features, built-in sensors such as high resolution cameras and GPS modules can be accounted for. Most of these features can be used to collect information relatively to the surrounding world, such as quotidian activities, that can be later used for entertainment or functional purposes. Normally, most of the smartphone applications, provided in the market, are used with the objective of entertainment or social connectivity. Nevertheless, these mobile systems have the potential to run applications for purposes other than entertainment, such as applications that can improve someone's life by using them. Initially, this Master's thesis work aimed to conceptualize the idea of using a smartphone to visually track an object selected by the user within a scenario captured by the smartphone's camera. As the name of the dissertation implies, the project pretends to track objects using a mobile device. Later on, there was a slight modification in the project's objective although the structuring goals persisted. In the end, the main goal of the project was the development of an application capable of detecting and tracking people for surveillance purposes, which was a feature included in a project called SAFETY being developed by Fraunhofer Portugal Research Center for Assistive Information and Communication Solutions (AICOS). The development of this dissertation was carried out under the scope of a Master's Thesis project for the Integrated Master in Electrical and Computers Engineering at Faculty of Engineering of University of Porto and in association with Fraunhofer AICOS Portugal.

1.1 Context

Nowadays, most surveillance systems existent in the market have several issues with their implementation. Firstly, the tracking techniques applied by most of these systems rely on background subtraction techniques, which imply the need of these systems to have a static environment for the tracking algorithm to function properly. This inability presented by the surveillance system does

not allow the installation of these equipments in mobile environments, leading to the usual installations to be in building infrastructures. Another negative point of these systems is the necessity of using expensive equipment in their installation.

In order to counteract those systems' limitations, it is necessary the creation of an inexpensive and mobile system capable of tracking objects in dynamic backgrounds. This new system would be applicable in mobile systems, such as cars and people, adding new type of scenarios where security forces could apply surveillance measures. Finally, this contribution could improve the work of security forces in their daily duties.

1.2 Objectives

At the present time, some security forces around the world have implemented recording cameras in their vests. The objective of these cameras is to document the scenarios and activities performed by these entities in special situations. Later, these recordings are analyzed by specialist teams to determine the causality of the actions performed by the individuals involved. One of the downfalls of this system, is its non-automation, having the user to turn on the video recorder.

The SAFETY project, in which this dissertation is inserted, is based on a smartphone incorporated in a vest especially designed for the security forces. This concept has the objective of aiding the security teams to later analyse the elements existent in their shifts, such as their routes, the time spent in each section of their route and detecting and recording dangerous activities occurring.

Within the SAFETY project, this dissertation has the main objective to automate the system already implemented by some security forces, as described before, by automatically detecting and tracking people in the video capture by a smartphone. Besides, it provides specific information in the video capture aiding the future analysis by the entities.

To accomplish that goal, it is necessary to develop a real-time visual tracking algorithm combined with a people detector. Its implementation was done in an Android environment with the help of the OpenCV ¹ library, for image processing purposes. Initially, the program was implemented in C++, with the aid of OpenCV, in Linux and later ported to Android via its Native Development Kit ² (NDK).

In order to evaluate the performance of these algorithms, a performance test is done using the BoBoT [1] dataset and applying the method of overlap, in which the overlap between the groundtruth data from the dataset and the algorithmic output is calculated, to measure the effectiveness of the algorithm.

A complete framework based on the algorithms developed and tested is available, setting a base for future works to improve the results obtained.

¹OpenCV: <http://opencv.org/>

²Android NDK: <https://developer.android.com/tools/sdk/ndk/index.html>

1.3 Contributions

From the developed work in the scope of this dissertation, several contributions have been made, such as:

- Development of a tracking framework for an Android application
- Study and implementation of TLD [2] and CMT [3] tracking algorithms in C++ using the OpenCV API in a host and Android platform
- Benchmarking of results from each algorithm by running tests using the BoBoT [1] dataset and applying the overlap method to determine their performance.

1.4 Structure

After this introductory chapter, Chapter 2 a literature review of the visual tracking computer vision algorithms is presented, as well the libraries normally used for the implementation of this kind of applications. In Chapter 3 an overview of the system's framework is presented a study of the platforms used and the study and implementation of the algorithms used for the purpose of this dissertation. Chapter 4 describes the evaluation method and dataset used for performance metrics of the system. Finally, in the last chapter the work developed is discussed and the dissertation main conclusions and future work are presented.

Chapter 2

State of the Art

In this chapter it will be discussed and analysed some of the algorithms found in the literature for computer vision systems that are the foundation for most of the video tracking systems used nowadays. The computer vision software will be analysed and subsequently the more appropriate choice will be presented.

2.1 Computer Vision in Object Tracking

In the last few years with the increase of robotic systems and artificial intelligence, the field of computer vision has upscaled significantly improving areas such as pattern recognition. The number of applications that these types of systems can improve are significant. Process control, event detection and automatic inspection are some of the applications that computer vision is able to perform leading the field of robotics and automation to a brighter future.

The event detection feature is the area where the topic of this dissertation, surveillance and object tracking, mostly fit in. Works in that area will be exploited to provide background information as well the state of art algorithms that are used in today's software. The discussion will be made upon tracking algorithms, such as Kernel-based Tracking, Point-tracking [4] and local features algorithms [5].

2.1.1 Kernel-based Tracking

Yilmaz *et al.* [4] presented a work explaining kernel tracking algorithms. The application of a kernel tracker implies a computation from two consecutive frames of the motion model obtained from the specified object. The object motion is described as the object's position variation with reference to time. Usually it is outlined in the form of a motion vector that interprets the transformation between two images or like Stiller *et al.* [6] referred to as parametric object motions. Within this category there is a subcategory built on templates and density-based appearance models that rely respectively in the mean shift [7] and KLT algorithm [8].

2.1.1.1 Mean Shift

In computer vision, mean shift algorithm is a nonparametric clustering technique that works without previous knowledge of the number of clusters and does not inhibit the format of the clusters [9]. According to Comaniciu *et al.* is possible to obtain the multivariate kernel density estimate using the following formula, given it a set x_i with n points on a d -dimensional space R^d :

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right) \quad (2.1)$$

with kernel $K(x)$ and windows radius h .

In order to achieve the mean shift vector it is necessary to perform the gradient of the density estimator 2.1 present next:

$$\begin{aligned} \nabla f(x) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (x_i - x) g\left(\left\|\frac{x-x_i}{h}\right\|^2\right) \\ &= \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right) \right] \left[\frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x \right] \end{aligned} \quad (2.2)$$

Based on the equation 2.2 it is possible to verify that the second term of the equation is the mean shift vector:

$$m_h(x) = \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x \quad (2.3)$$

Basically the mean shift algorithm aims to locate the maxima of the density function given a set of points. It is based on an iterative process that shifts the center of the kernel to the mean position of the points contained in the cluster, refereed to as the estimated maxima. The shifting process is determined by the mean shift vector 2.3 that stops when there are no more points adequate to include.

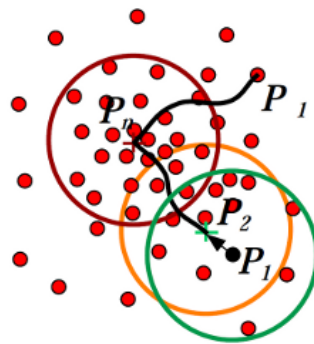


Figure 2.1: Mean Shift analysis [10]

Figure 2.1 exhibits the iterative shifting process, it is possible to observe the cluster's behaviour as it starts from the initial point until it is impossible to agglomerate any more points.

Based on this approach, Comaniciu *et al.* [11] added to the computer vision field a new tracking algorithm using the mean shift theory explained *a priori*. The Bhattacharyya coefficient [12] is applied to find the target location depending on a certain feature. Normally, that feature is either color and/or texture of the object selected and it is retrieved from its histogram. Starting from the general form of the Bhattacharyya coefficient 2.5 it is possible to find the discrete location that lead Comaniciu *et al.* to deduce the following equations where q_z is the density function and \hat{q}_z is the discrete density function of the target:

$$\rho(y) \equiv \rho[p(y), q] = \int \sqrt{p_z(y) q_z} dz \quad (2.4)$$

$$\hat{\rho}(y) \equiv \rho[\hat{p}(y), \hat{q}] = \sum_{u=1}^m \sqrt{\hat{p}_u(y) \hat{q}_u} \quad (2.5)$$

$$d(y) = \sqrt{1 - \rho[\hat{p}(y), \hat{q}]} \quad (2.6)$$

Resorting to the equations already explained the tracking algorithm is built on the following steps:

- **Step 1** — Initialize the target's location and its evaluation 2.6.
- **Step 2** — Derive the weights achieved after modifying the Bhattacharyya coefficient for distance minimization.
- **Step 3** — Derive the new target's location using the mean shift vector 2.3.
- **Step 4** — Update and evaluate the target's location 2.6.
- **Step 5** — The process stops if the difference between consecutive positions is less than a predefined threshold.

In summary, the procedure consists in minimizing the Bhattacharyya distance in reference to the previous location estimate that will be set as the new position of the target model in the current frame. This algorithm is implemented in OpenCV ¹, a computer vision library provided by Intel, as CAMSHIFT and which results can be seen in the Figure 2.2.

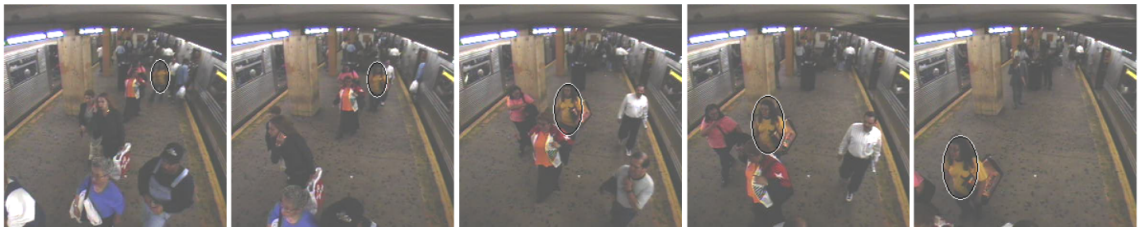


Figure 2.2: Mean Shift tracking results [11]

¹OpenCV: <http://opencv.org/>

2.1.1.2 Kanade-Lucas-Tomasi Feature Tracker

Feature extraction is an area of computer vision that its objective involves building features from sets of data allowing the creation of simplified ways of analysing complex data. KLT feature tracker is an approach which essentially uses spatial intensity information to find the best position for the features identified *a priori*.

Firstly, Lucas and Kanade [13] introduced the idea to weigh the gradients using an approximation to the second derivate of the image for a local search. Later, Tomasi and Kanade [14] concluded that by tracking features that are suitable for the tracking algorithm it would improve the algorithm's technique.

In an one-dimensional case the registration algorithm relies on an approximation of the gradient of the image, obtaining the displacement between two images.

$$F'(x) \approx \frac{G(x) - F(x)}{d} \Leftrightarrow d \approx \frac{G(x) - F(x)}{F'(x)} \quad (2.7)$$

$$d \approx \frac{\sum_x F'(x) [G(x) - F(x)]}{\sum_x F'(x)^2} \quad (2.8)$$

Nevertheless, 2.7 approach is inaccurate if the displacement d is too large. Therefore, it is considered a local area surrounding a feature 2.8 to improve that inaccuracy and the calculation of the displacement is done by using an iterative search style.

After the features registration, they would be selected for tracking if the eigenvalues from the gradient matrix were bigger than a predefined threshold, $\nabla d = e$. The results obtained from this method are presented in the Figure 2.3:

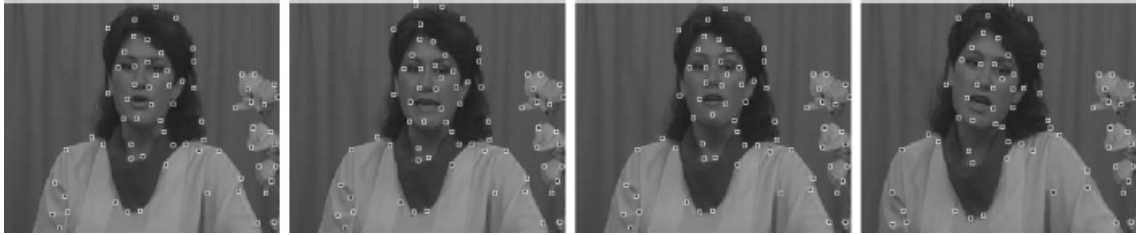


Figure 2.3: KLT tracking results [4]

This algorithm is widely used for image alignment or feature position tracking in computer vision. OpenCV has implemented it using "good features to track" and optical flow algorithm based on the KLT, creating an array of options for the user to choose the most appropriate algorithm for its project.

2.1.2 Point-tracking

Yalmaz *et al.* [4] presented that strategy used for tracking sets upon the correlation of objects modeled by points across the frames. There are difficulties concerning those algorithms because

occlusions, objects out of bounds and detection failures can induce into a bad correspondence between different frames. This can be split in two different categories such as deterministic and statistical methods. The statistical methods rely on two of the most used algorithms in computer vision as filtering algorithms named as Kalman Filter [15] and Particle Filter [16].

2.1.2.1 Kalman Filter

Kálmán [15] developed an algorithm named after himself which utilizes measurements containing noise and statistically produces estimates of the system state more accurate than capturing single measurements instead. This strategy is most fitted when the system is linear and assumes a Gaussian distribution.

This algorithms consists in two steps, the prediction and correction or update step. Respectively, the prediction step occurs when the filter estimates the current state variables and the update step exists when there is observable measurement data updating the estimates achieved while using the prediction step. Subsequently, those steps are modelled using the following equations [17]:

$$\text{Predict} : \hat{x}_{k|k-1} = F_k \hat{x}_{k-1} + B_k u_k \quad (2.9)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad (2.10)$$

$$\text{Update} : \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k \quad (2.11)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \quad (2.12)$$

Where the \tilde{y}_k is the measurement residual, K_k the optimal Kalman gain, $\hat{x}_{k|k}$ is the *a posteriori* state estimate and $P_{k|k}$ is the *a posteriori* error covariance matrix. A model of the Kalman algorithm is presented in Figure 2.4 where it demonstrates how the prediction and update steps interact with each other in order to validate it.

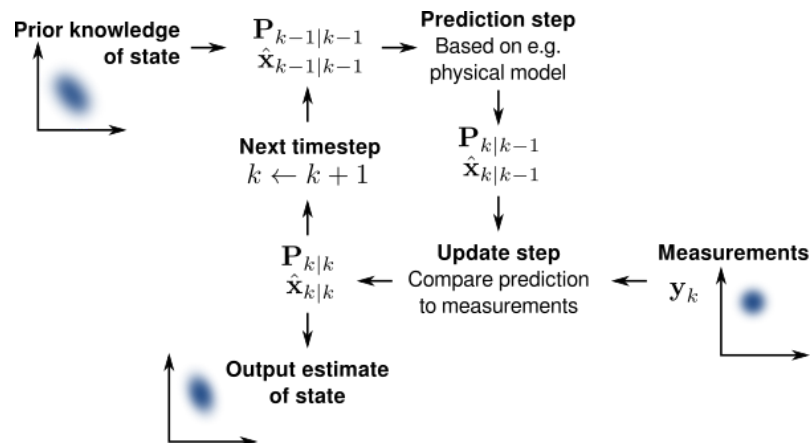


Figure 2.4: Kalman filter model

The Kalman filter has wide applications in the area of technology. Mostly its application is in

control systems but has increased its importance in the computer vision field as a filtering algorithm. As explained before, it has the power to process all the measurements taken and statistically predict according to Gaussian distributions the system's state variables. This design has improved dramatically tracking techniques by improving them during the occurrence of occlusions or mismatches of the tracking algorithm. Achieving this, most of the computer vision software have implemented it in their APIs facilitating the use of this technique.

2.1.2.2 Particle Filter

The particle filter is a method utilizing a Monte Carlo technique to solve the problems of the state estimation [16]. This method is also recognized as a filtering technique and condensation algorithm. The essence of this algorithm is to use sets of random data, also described as particles, with associated weights to express the posterior density function required. Additionally, these weighted particles are computed in order to obtain the estimate states needed. If there is a significant increase in the number of particles, there is a need of using the Monte Carlo characterization to represent the probability function and achieve a solution approaching the optimal Bayesian estimate [18]. The particle filter overcomes Kalman filter's limitation, the inefficiency of estimate states when the variables do not follow a Gaussian distribution [4].

One of the most used algorithms in particle filtering is the purported Sequential Importance Sampling (SIS) algorithm, that utilizes a resampling step at each instant [18]. SIS exploits a density which represents one type of density that cannot be computed and it is named importance density or posterior density in this case. Based on those characteristics samples are drawn from it instead from the actual density.

The posterior density can be approximated in discrete terms at t_k by:

$$\pi(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^N w_k^i \delta(x_{0:k} - x_{0:k}^i) \quad (2.13)$$

Where δ is a dirac delta function, $x_{0:k}^i$ are the particles with their w_k^i weights and $x_{0:k}$ the set of all states up to t_k .

Assuming a Markovian process, the posterior density can be modelled as [19]:

$$\pi(x_k|z_{1:k}) \approx \sum_{i=1}^N w_k^i \delta(x_k - x_k^i) \quad (2.14)$$

SIS algorithm has one setback pointed as the degeneracy phenomenon. It happens when all particles but one will have negligible weight causing the update of the particles to have a great computational cost. In order to overcome this problem a bigger number of particles or resampling techniques is necessary. The Sampling Importance Resampling (SIR) is one of these techniques which involves a mapping of random measure $\{x_k^i, w_k^i\}$ into a random measure $\{x_k^*, N^{-1}\}$ with uniform weights, which are applied indistinctively at every instant t_k [18, 19].

The algorithm SIR consists in three steps and graphically demonstrated in Figure 2.5.

- **Step 1** (Prediction) — Calculation of the particles weights using the likelihood function.
- **Step 2** (Update) — Normalization of the particle weights by calculating the total weight.
- **Step 3** (Resampling) — Resampling of the particles by assigning samples and weights aided by the computation of the cumulative sum of weights.

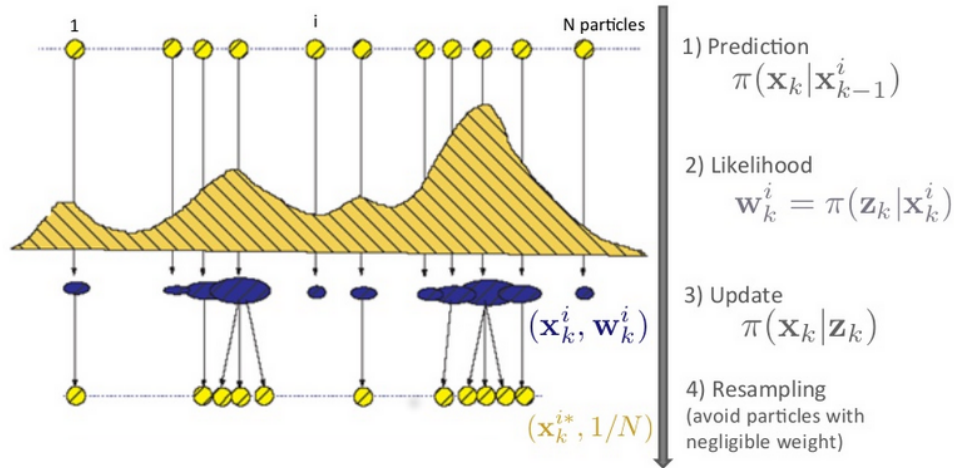


Figure 2.5: SIR algorithm model [20]

Although this method can overcome the degeneracy problem, its use can create a loss of diversity leading to a series of repeated particles in the resultant sample.

One use for this method is the condensation algorithm [21]. Condensation is a probabilistic algorithm capable of detecting and tracking moving objects' contours in a cluttered environment and is an application of SIR. In terms of applications it is usually used for tracking but can also be applied to recognize human gestures, allowing the machine to capture and recognize simple human gestures. OpenCV has an implementation of this algorithm that can work as the condensation algorithm or a particle filter when a likelihood function is created in order to track the particles under its effect.

2.1.3 Local Features

In computer vision, Grauman *et al.* [5] defines that local features have the function of producing an efficient representation of the image structure to allow matching between different images. Basically, the objective of these methods is to acquire a set of local measurements able to represent the image and its structure in form of data, they can also be called as keypoints. The process to obtain local features for matching consists in two phases: feature extraction and feature description. Respectively, the feature extraction should be a process repeatable, precise and distinctive in order to achieve the same features for similar objects' structures in different images while the feature descriptor is a technique used to compare the features obtained using the feature extraction. In other words, the functionality of the descriptor is to assign a numerical description to

the area of the image that the feature refers to and they should be independent of feature position, robust against transformations and scale independent. Once the descriptors are calculated it is now possible to compare features obtained in different images to see if they match. The mechanism explained previously is illustrated in Figure 2.6 and implements the following algorithm [5]:

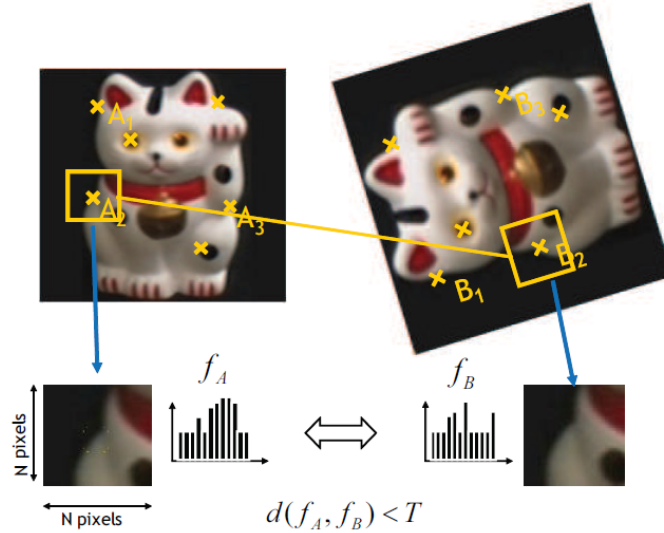


Figure 2.6: Local features recognition procedure [5]

- Feature Extraction
 1. Detection of a set of keypoints.
 2. Definition of a region surrounding each keypoint in a invariant way.
 3. Extraction and normalization of the region content.
- Feature Description
 1. Computation of a descriptor from the normalized region.
 2. Matching the descriptors obtained.

In addition, two algorithms that recur to these type of techniques will be discussed: SIFT [22] and HOG [23], that are largely used in the industry nowadays.

2.1.3.1 Scale Invariant Feature Transform

The Scale Invariant Feature Transform is a combination of a DoG detector and a descriptor of the same name [22, 24]. Nowadays, SIFT algorithm has multiple applications like object recognition and identification, 3D modeling and video tracking. The University of British Columbia has the algorithm patented in the United States of America [25].

Lowe [24] demonstrated the scale-space Laplacian can be approximately achieved using DoG. Essentially, for the keypoints detection to happen the image is convoluted with Gaussian filters

using different scales. Successive images are later subtracted by a scale factor of k . This technique is called the Difference of Gaussians and its image $D(x, \sigma)$ can be modeled by:

$$D(x, \sigma) = (G(x, k\sigma) - (x, \sigma)) * I(x) \quad (2.15)$$

The keypoints are identified as local minima/maxima on the DoG images obtained. This step requires the examination of each pixel in the image with its eight neighbors in the same and neighboring scales to verify if it corresponds to a maximum or minimum in comparison with the other pixels in order to be selected as a candidate keypoint.

After the keypoints detection step it follows the SIFT descriptor. The objective of this descriptor is to accomplish robustness in variations of lighting and position by storing the image information in a localized set of gradient orientation histograms. Initially, the descriptor starts by sampling the surrounding keypoint area to determine which level of Gaussian blur it represents. This sampling is computed utilizing a 16x16 grid to cover the interest region. On each sample, a 4x4 grid is introduced representing gradient orientating with 8 bins, each which weight is determined by a Gaussian function with a σ of half the width of the descriptor window and corresponding pixel's gradient magnitude. In the end, the descriptor is a vector containing all these histograms values achieving a 128 (4x4x8) dimensional feature vector. To finalize the vector is normalized to unit length as an illumination normalization for the extraction procedure. To compensate the effects of non-linear illumination changes a threshold is applied to a maximum value of 0.2 to the vector and again normalized.

SIFT is one the most used feature descriptors used. Most of computer vision software have developed its functionality even though it is patented. This is done because of its major potential comparing to others descriptors. Although it is not ideal and requires a large amount of computational power, it is still one of the algorithms that presents best results in complex tasks containing rotation, scaling and motion blur.

2.1.3.2 Histogram of Oriented Gradients

The Histogram of Oriented Gradients is a feature descriptor introduced by Dalal *et al.* [23] where local object appearance and shape in the image is defined by intensity gradients or edge direction distribution. The simpler method to achieve the gradient computation is to employ the 1D centered point discrete derivative mask in the horizontal and vertical directions. It implies filtering a grayscale image with the following filter kernels shown:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad D_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \quad (2.16)$$

Given an image I it is possible to obtain its derivatives I_x and I_y by performing a convolution between I and the filter kernels 2.16. Using the derivatives it is able to obtain respectively the

magnitude and orientation of the gradient:

$$|G| = \sqrt{I_x^2 + I_y^2} \quad (2.17)$$

$$\theta = \arctan \frac{I_y}{I_x} \quad (2.18)$$

Resorting to the previous calculations, the HOG algorithm implementation is achieved by dividing the image into cells and performing within each cell a histogram of gradient directions or edge orientations. It implies casting a weighted vote from each pixel for a orientation-based channel. Dalal *et al.* defined rectangular cells with histogram channels spread from 0 to 180 or 360 degrees depending, respectively, if the gradient is unsigned or signed and the contribution for the vote weight relies majorly on the gradient magnitude 2.18.

The combination of these histograms are the representation of the descriptors vector. The performance can be augmented by calculating the measure of intensity in a block and using those values to normalize all cells within the block. The descriptors block can be of two geometric types: R-HOG and C-HOG. Normally, they are square grids and are represented by the number of cells, number of pixels per cell and the number of channels per cell histogram. The normalization process can be produced with different methods, being the most used the followings:

$$\text{L2-norm} : f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}} \quad (2.19)$$

$$\text{L1-norm} : f = \frac{v}{\|v\|_1 + e} \quad (2.20)$$

$$\text{L1-sqrt} : f = \sqrt{\frac{v}{\|v\|_1 + e}} \quad (2.21)$$

Where v is the non-normalized vector accommodating all the histograms in a certain block. This normalization process enables a better invariance to illumination changes.

The HOG algorithm with a SVM classifier is one of the most used person detector in computer vision software. One of the topics of this dissertation resides in surveillance issues so the ability of detecting people is one of the subjects issued and this feature descriptor algorithm conjoined with a linear SVM classifier using the INRIA dataset [23] is capable of obtaining good results. The diagram in Figure 2.7 explains how the process from the input image until the person detection is made:



Figure 2.7: Overview of HOG feature extraction and object detection diagram [23]

There are programs that uses it in order to track people in video streams. Nevertheless, the

APIs provided from the most used computer vision libraries have a high computational cost because it is based in a sliding window method resulting in a slow processing speed when dealing with higher resolution images.

2.1.4 Summary

Tracking systems are very challenging pieces of software. The amount of information contained within an image is enormous and the ability of gathering the relevant information is a challenge. The algorithms presented can work on their own but when combining some of them better results can be achieved. For example, the utilization of filtering algorithm as the ones presented in the section 2.1.2 if combined with mean shift algorithm can improve significantly its performance. The challenge of occlusion and mismatching can be partially resolved using the Kalman or particle filter because they have the ability to predict the next state and therefore guide the tracking agent to an estimate position improving the tracking system by eliminating some variables within the image. Another problem is the computational cost that usually this systems have creating problems when trying to processing real time video streams or operating with hardware of reduced processing power. In that sense, the computer vision area is progressing to a domain where is trying to reduce its computation cost while having the same or better results obtained in the past.

2.2 Computer Vision Software

The ability of producing in a faster and reliable way computer vision software comes from the application of computer vision libraries developed and which contain most of the state of art and basic algorithms created in the history of computer vision. Currently, OpenCV and MATLAB's Computer Vision System Toolbox are the most used in the industry and for investigation purposes.

OpenCV ² is a software library developed for real-time computer vision by Intel in 2000. It is an open-source cross-platform library created for free use. OpenCV possesses multiple modules containing more than 350 algorithms related to computer vision and is developed in C/C++ allowing a greater processing speed that enables real-time image processing. Since its creation it has been given support in different programming languages as well in different operation systems such as Windows, Android, iOS, Linux and OS X.

MATLAB ³ is a high-performance interactive software aimed to perform numerical calculation and a high-performance language for technical computing developed by MathWorks. MATLAB software allows numerical analysis, matrix calculus, signal processing, plotting of functions and data, creation of user interfaces and the ability of multi-platform interface with different languages programs. Basically, MATLAB's basic information element is a non-dimensional matrix. Using it, the systems ability to solve numerical programs is way faster than to transcript it to another programming language. The Computer Vision System Toolbox provides algorithms, functions and apps capable of performing computer vision tasks defined by the user.

²OpenCV: <http://opencv.org/>

³MATLAB: <http://www.mathworks.com/products/matlab/>

Having introduced the software available there is a need of discussing which of them is the most reliable in terms of image and video processing to implement the software proposed for this dissertation. According to the information retrieved from Fixational [26] on both programming environment and tools we were able to produce the benchmark presented in Table 2.1.

Table 2.1: Computer Vision software performance benchmark (higher is better)

Performance Features	OpenCV	MATLAB
Ease of Use	2	5
Speed	5	1
Resources Needed	5	2
Cost	5	4
Development Environment	3	4
Memory Management	2	5
Portability	4	2
Development of Useful Programming Skill	4	2
Debugging	3	5
Help And Sample Code	5	5

From the scores obtained from the Table 2.1 it is possible to determine that OpenCV as a slight edge over MATLAB for this objective. MATLAB has its specific language that is easy to use and learn, has an integrated memory management software and an enormous array of documentation and sample code but lacks execution speed because of its high computational cost. OpenCV, on the other hand, having as main programming language C/C++ increases the difficulty on issues related to debugging, memory management and others, but wins against MATLAB because of being open-source, ability of embedding the developed code into applications and its processing speed. Those features come from the key idea that OpenCV was entirely made for image processing use, having its functions and data structure specially design for the purpose of processing images.

On conclusion, the software chosen for this dissertation was OpenCV mostly because of its capability of embedding the code created into an application, its major processing speed comparing to others software and the key factor is the ability to develop applications for Android environment. The dissertations main topic is the creation of a tracking application for mobile platforms and OpenCV is the only software capable of fulfilling that need.

2.3 Conclusions

Computer vision systems for non-static surveillance or mobile adaptation are currently areas of research in progress. There has been a great need in their development since all of surveillance systems are static and the mobile integration of computer vision systems have not yet been completely developed. At a scientific level, the algorithms analysed *a priori* can serve as the base of

an algorithm capable of implementing a dynamic tracking system. The algorithms presented in this chapter are used in the foundation for a wide variety of image processing algorithms and their study was needed to obtain more knowledge to better face the problem of this dissertation. Nevertheless, the ability to integrate into a mobile device is one of the big challenges faced because of processing power. The development of these algorithms normally use a great amount of computational power and if they see being used in a desktop environment it is easier to assure its real-time performance because of the system specifications as having a multiple core processors and graphic processing units. OpenCV in speed comparison to Matlab has a better performance and therefore for smartphone applications it is more suitable to use OpenCV because of the processing issues that mobile platforms have.

Chapter 3

Real-time Tracking System of Unknown Objects

In this chapter an overview of the system, software environment, algorithms used and application development is presented. From a technological perspective, the process since the image capture until the final output of the tracking application will be explained. In detail, the system's framework and how it is able to exploit the best practice for this application will be explained. Conceptually, the algorithms used will be examined and translated into a programming environment suitable for the purpose of a mobile platform integration. In this application, the portability between platforms is crucial because the creation of the system is more efficiently developed in a controlled environment that permits testing with different tracking datasets allowing to obtain quantitative results. The application is then transferred to the final module.

3.1 Architecture

The smartphone device should be used to capture video streams from quotidian activities and process the information received in real-time to track the object selected *a priori* in the frames received from the camera.

Therefore, the architecture of the system for tracking single unknown objects selected by the user must be done according to the next steps:

1. Video capture
 - Smartphone camera for recording live video streams.
2. Definition of the region of interest
 - After the initialization of the video stream a bounding box must be defined to select the object of interest selected by the user.
3. Computation of the tracking algorithm

- Having selected the ROI the algorithm should process the information gathered from the object selected and the frames captured from the camera providing enough data capable for the tracking process to perform across all video frames.

4. Visualization of the tracking algorithm result

- There is the need to draw the bounding box obtained from the object tracked in the current video stream to assure that the user visualizes the results retrieved from the algorithm.

The previously identified steps indicate how the system should be solved in order to address the needs of the project. The specification that a predefined object appearance and data is not provided beforehand the start of the video capturing makes the step 2 of rather importance. Its importance comes from the need to obtain information about the object to track and consequently the necessity of having to select it from the video itself. Only then it is possible to process whatever tracking algorithm for its specific purpose. Without the objects' information such as features [5], motion models [4], histograms and others it is impossible to acquire the necessary information from the video because there is no point of comparison and matching between the frame and the object. Another important specification from the point of view of the user is the visualization of the object, step 4, when the algorithms is being processed. It allows the user to acknowledge if the algorithm is processing the information correctly and if its behaviour is adequate for the object of interest selected.

On the other hand, most of the surveillance systems are automated and do not require the user interface to detect and select the objects to be tracked. In that sense, it is necessary to use an algorithm capable of delivering the detection of the objects to be tracked later using the same model presented before. The system framework can be translated as a loop and in Figure 3.1 it is possible to verify that feature.

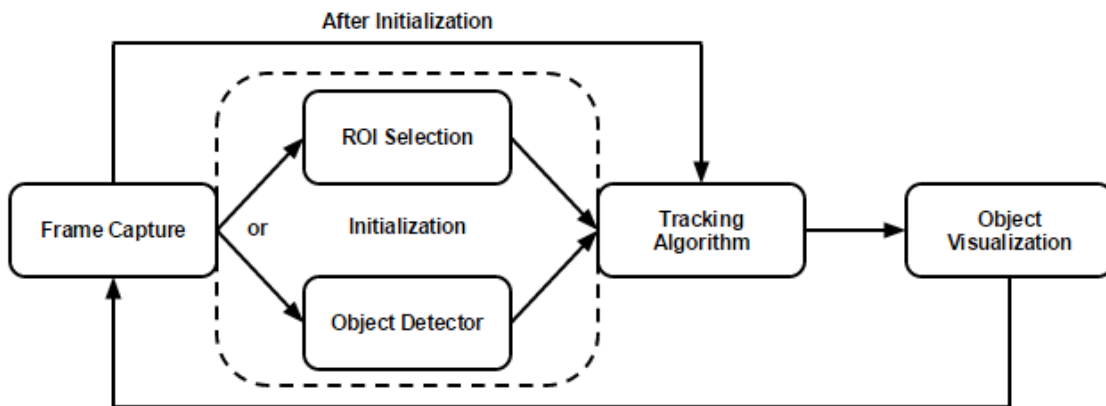


Figure 3.1: System framework diagram

Firstly, in order to facilitate the use of datasets to test the performance of the algorithms implemented, it will be chosen the manual initialization process. This process has the goal to enable the choosing of the initial region of interest provided from the dataset to be able to execute the evaluation method. If the algorithms present the results expected, the final system will be presented with the people detector implemented for the selection process, as the final version of this project.

3.2 Software Platforms and Portability

The system's final stage must be a mobile application for a smartphone. This specification implies the development of software capable of being easily transferable between UNIX based software and the Android environment. The software's purpose is image processing, therefore both environments must be able to run and make use of OpenCV library. Android is one of the mobile phone operating systems that has an OpenCV library functionally developed making it possible to create applications for image processing more easily. Also Android provides coding techniques that allow reusing C/C++ code programmed in the desktop version and transfer it with some adjustments to the Android environment without great effort. Figure 3.2 demonstrates the process of the desired system portability between platforms.

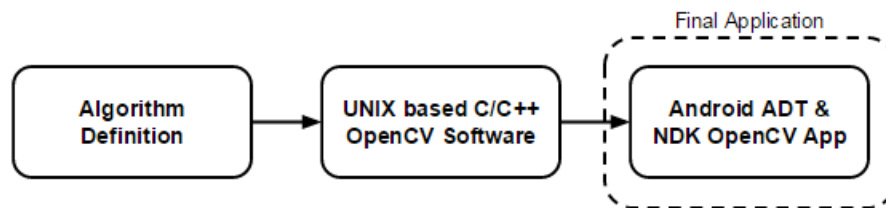


Figure 3.2: Software project flow

This section will explain the software's behaviour in each platform and the techniques used to allow the transfer between them.

3.2.1 UNIX based Software

The Linux OS has the ability of utilizing the OpenCV library easily while working on the Eclipse IDE ¹. This IDE has the ability of creating C/C++ projects and then include the necessary OpenCV libraries by linking and including them in the project itself. That capability facilitates the software development as it allows to effortlessly use the libraries provided by OpenCV. The array of modules implemented by the version 2.4.11 of OpenCV ² are as it follows:

- **core** — module containing the basic functions and data structures that creates the foundation of OpenCV and are used by the other modules.

¹Eclipse: <https://eclipse.org/>

²OpenCV API Reference: <http://docs.opencv.org/modules/refman.html>

- **imgproc** — module based on image processing techniques like image filtering presented in the section 2.1.2, image transformation, histograms, etc.
- **video** — module which includes tracking algorithms, motion estimation and background subtraction techniques used for video analysis.
- **calib3d** — module containing basic camera calibration techniques and multi-view geometry algorithms for correspondence or reconstruction of objects.
- **features2d** — module based on feature detection and description techniques as presented in section 2.1.3.
- **objdetect** — module used for detection of objects using specific classifiers, like the people detector explained in the section 2.1.3.
- **highgui** — module capable of interacting with video and image codecs and application of user interface techniques.
- **gpu** — module with the GPU-accelerated algorithms of some of the other modules presented in the OpenCV library.

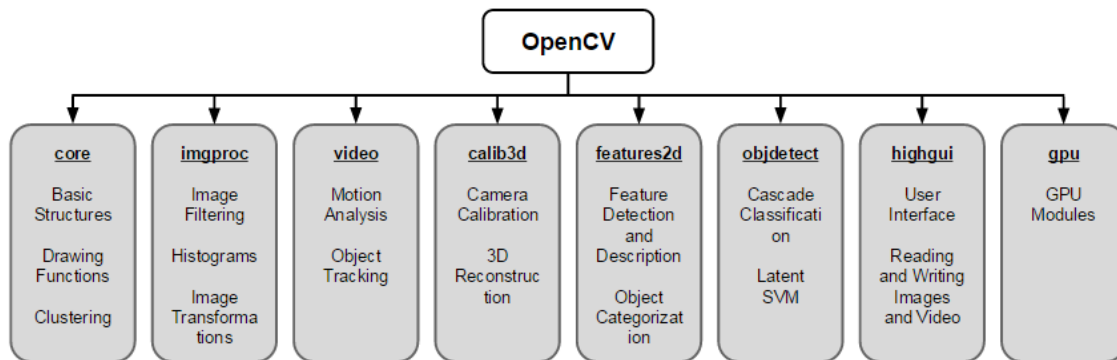


Figure 3.3: OpenCV modules

Figure 3.3 shows the modules and what type of processes they translate into the programming world. Also these modules are responsible for all the algorithms present in the OpenCV library. In this dissertation, the most used modules were the core, imgproc, objdetect and highgui.

In terms of API for C/C++ programming, the OpenCV provides automatic memory management and allocation of output data. When handling memory the system has destructors that deallocate the memory buffers used when using the data structures present in the library when necessary. Regarding output memory the system deallocates the memory automatically while it allocates and reallocates the memory for the output arrays with reference to the size of the input arrays of the function and some other parameters defined by OpenCV.

With respect to multi-threading, OpenCV provides a fully re-enterable implementation. This means that different threads can call the same function, method and class. It creates a faster processing because it can run multiples instances of the same function in different threads allowing parallel programming which leads to faster processing times if the hardware has the correct specifications.

Concluding, it is possible to affirm that the development of the software in these respective environments creates a suitable hypothesis considering speed, ease of use and the functionality intended for the application itself. In those terms the diagram presented in Figure 3.4 can transcribe how the system is projected in a UNIX based environment.

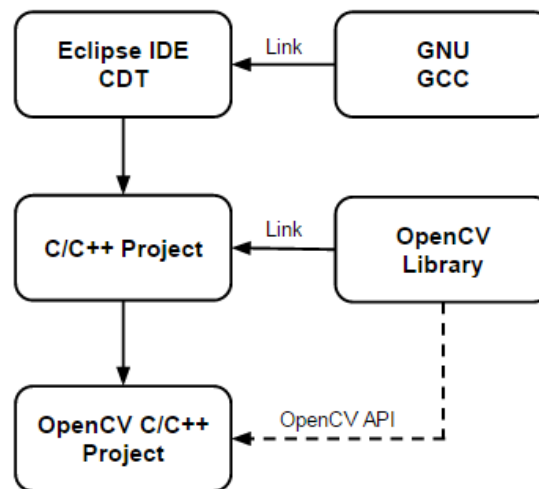


Figure 3.4: C/C++ project flowchart diagram

3.2.2 Android OS

The computer vision software for mobile applications is far from being ideal. Nevertheless, OpenCV has adapted its library to the mobile environment such as iOS and Android. The environment used in this dissertation is Android and OpenCV ensures its collaboration with a library named OpenCV4Android [27] for Android Studio and SDK tools.

OpenCV4Android SDK package enables the use of OpenCV library in the development of Android applications. The package itself contains OpenCV libraries for Android, an Android library for Eclipse projects providing OpenCV Java API, OpenCV C++ headers and native Android libraries and, finally, Android packages that should be installed on the target device to enable the OpenCV library. The package installed is OpenCV Manager API.

OpenCV Manager is an Android service that allows users and developers of OpenCV applications to benefit from:

- OpenCV applications apk-size more compact as a result of all applications using the same binaries and native libraries from Manager.

- Automatic updates and hardware optimizations enabled.
- Trusted OpenCV library source.

Resorting to these packages OpenCV provides a possibility to develop using two different techniques. The first one utilizes the OpenCV Java API and OpenCV4Android to create applications using Java programming language and the corresponding OpenCV API. Since the Android main programming language is Java, having the ability of incorporating OpenCV functions in the core code enables a more accessible coding and development. One of the problems concerning the use of the Java API is the incomplete implementation of OpenCV functions which leads to insufficient resources when developing some types of applications. Also, with the Java API, OpenCV C++ functions continue to be performed at the native level but a Java wrapper is used as an interface. The problem with the wrapper results comes with a performance shortfall from the JNI overhead. The Java Native Interface overhead occurs at the start and end of each function call. Therefore, this performance deficiency improves cumulatively with the number of OpenCV functions being made within the application.

The OpenCV Java API runs under Dalvik Java virtual machine which is a process virtual machine of Android OS that executes applications written for Android. The OpenCV Java API system is translated in the diagram of Figure 3.5. It exemplifies how the function calls are made using this method.

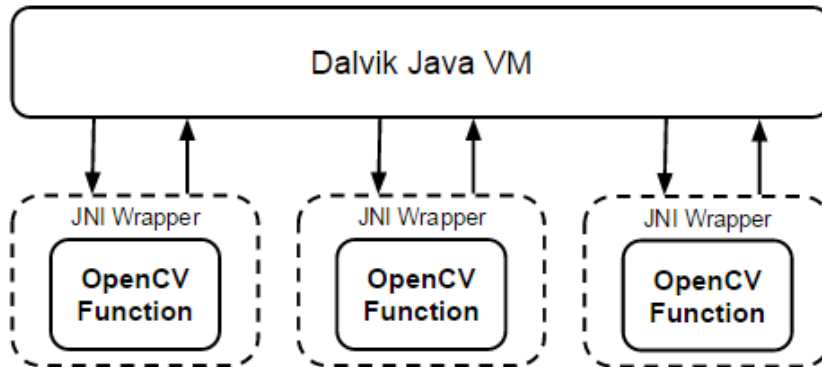


Figure 3.5: OpenCV Java API application structure

Figure 3.5 demonstrates an Android application calling three OpenCV functions per video frame. It also shows that each call translates into two JNI overheads, resulting in the addition of all which translates into the final result of six JNI overheads. The result is a linear overhead accumulation in respect to the number of function calls made.

The problem with this technique is that it affects the performance of the algorithm used and it does not follow the requirements of portability between the application created beforehand in an UNIX based software and the Android platform. Considering these obstacles, the second technique proves to be more useful for this purpose and it is achieved by using Android's Native Development Kit.

NDK is a set of tools used to embed C/C++ code into Android applications. Developers find it useful when it comes to apps portability between platforms, exploiting already developed libraries and overcoming performance issues presented by the Java API. Using this approach, most of the OpenCV functions code is written entirely in C/C++ with direct calls to OpenCV. Therefore, it is possible to embed all the OpenCV functions into one single C/C++ class which is called once per frame. In contrast to the Java API, this method only requires two JNI overheads per frame improving the performance of the software significantly and independently of the number of OpenCV function calls. This technique reduces the use of Java significantly, basically just being predominantly used in non-CV aspects of software like the GUI.

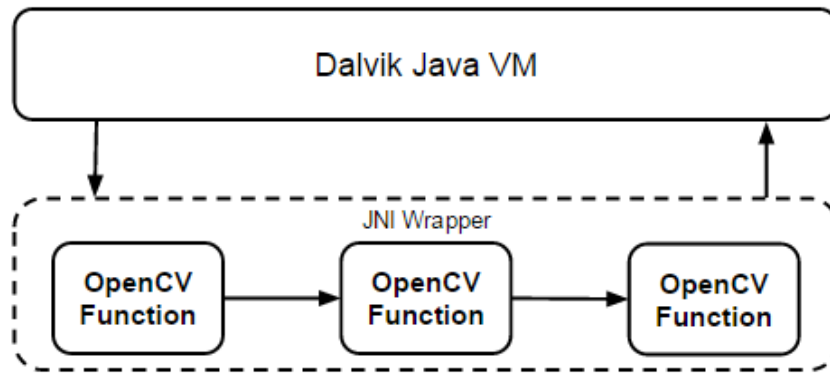


Figure 3.6: OpenCV application structure using NDK

Figure 3.6 in comparison to the Figure 3.5 allows the observation of the different software dynamics happening in both cases. In the NDK case, it significantly improves in terms of performance and ability to transfer the same application between platforms. Therefore, this method has the specifications desired for the project because it is possible to develop and test the algorithm implementation on a desktop platform, like the one discussed in 3.2.1, being later ported and implemented into an Android project.

In conclusion, the project work flow will have in the development and test stages of the algorithms implementation on a desktop platform such as Linux and later ported into a mobile platform with the aid of NDK from Android. This will speed up the process of porting the code to a mobile platform by facilitating the use of the implementation already done in C/C++ in Linux.

3.3 Object Tracking Software

Having previously discussed how the system's project work flow is designed and implemented, it is time to discuss the algorithms underlying the tracking software. In this section two algorithms implemented during the realization of this dissertation will be presented. Firstly, the theoretical elements and structures of each algorithm will be explained and later how they are implemented in the system. The algorithms presented are Tracking-Learning-Detection [2] and Consensus-based Matching and Tracking of Keypoints [3].

3.3.1 Tracking-Learning-Detection

Kalal *et al.* [2] proposed an algorithm for object tracking that exploits the tracking task into three steps: tracking, learning and detection. Basically, the tracking algorithm is performed over consecutive frames in order to follow the object after being localized by the detector. The detector has the function of pinpointing all the appearances observed and correct the tracker. In order to improve the detector return, a learning process that estimates its errors and updates to bypass the errors already presented before is used. The technique used for the learning step is based on a P-N learning method: P estimates missed detections, and N estimates false alarms. The TLD framework explained before is shown in Figure 3.7.

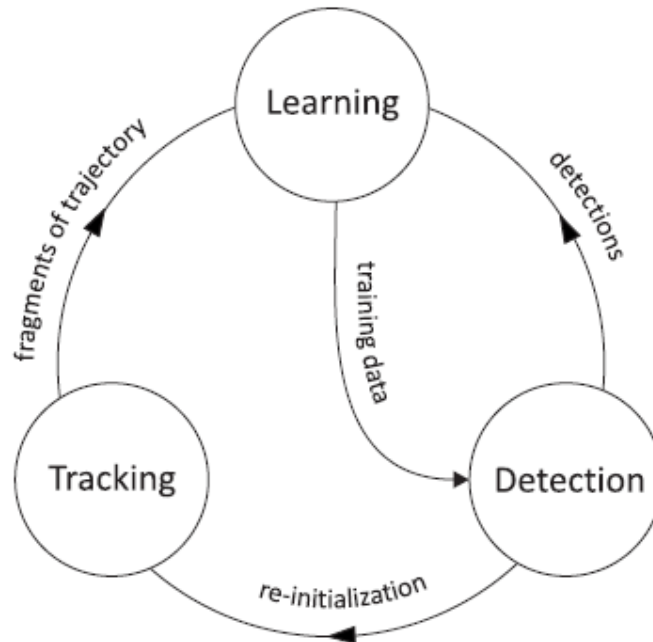


Figure 3.7: TLD framework diagram [2]

After the TLD algorithm by Kalal *et al.* was proposed some variations of the same algorithm with some optimizations have appeared, resulting in the achievement of better outputs. One of these variations is the algorithm presented by Nebehay *et al.* called OpenTLD [28]. Like the TLD algorithm, it is also divided into three phases but with fewer modification. Figure 3.8 shows that the initialization starts the learning process. Next, both Tracking and Detector steps start executing in parallel, achieving one final result that needs to be validated; if the validation is correct then the learning is accomplished.

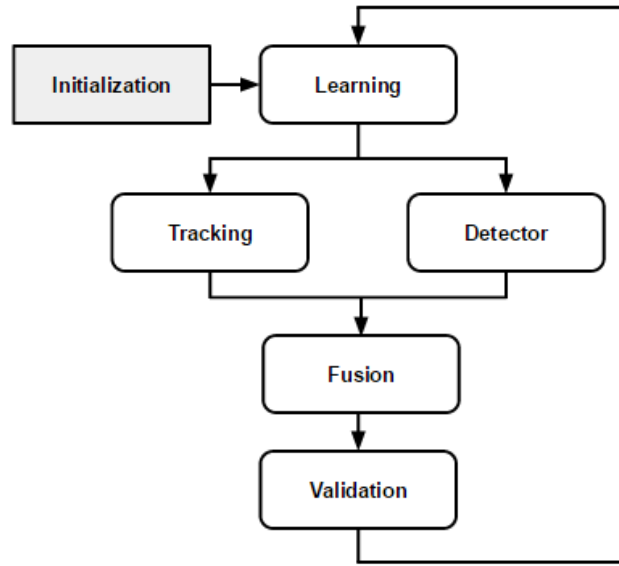


Figure 3.8: OpenTLD architecture

After the architecture description had been made, it is necessary to thoroughly explain each of the three major steps present. Therefore, the next sections will be referring how they are structured and work.

3.3.1.1 Tracking

The tracking method goal is to locate the position of the object in the current frame. Specifically, the method used is the recursive object tracking proposed by Kalal *et al.* [29].

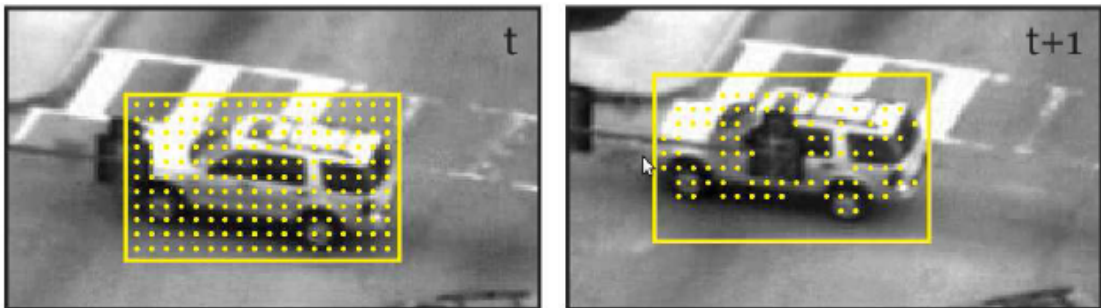


Figure 3.9: Recursive tracking method [29]

Figure 3.9 demonstrates the principle of recursive tracking. Firstly, in frame t an array of equally spaced points is constructed inside the bounding box. The optical flow [13] on each point is estimated and later the points expected to be inaccurate are filtered. This filtering process is done by applying a normalised correlation coefficient and measuring the forward backward error [29], which the last one measures the discrepancies between the trajectories performed forward

and backward by the tracking algorithm. In the frame $t + 1$, the remaining filtered points are exhibit. The remaining points are used to obtain a new bounding box if the average of all forward-backward errors measures is bellow a predetermined threshold. The new bounding box is then calculated using the transformation model presented by Kanal *et al.* [29].

3.3.1.2 Detection

The recursive tracker algorithm is only possible to be used while the selected object is visible in the image. Therefore, the process of detection is applied to re-initialize the recursive tracker to avoid tracking failures in case of occlusions because the model of the recursive tracker it is not able to maintain the object's model.

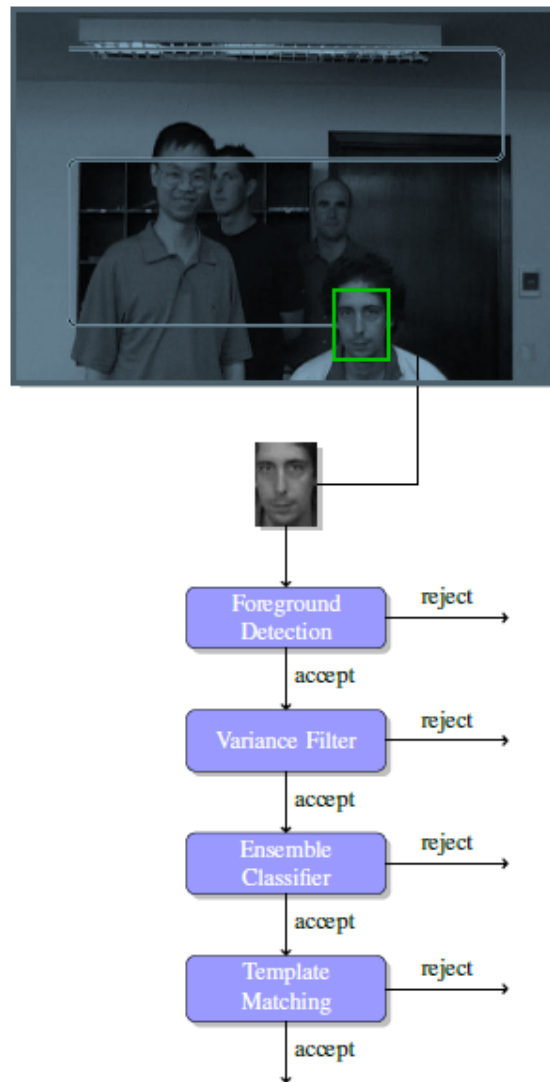


Figure 3.10: Detection cascade approach [28]

The object detector uses a sliding-window technique [30, 23]. The objective of this approach is to filter out most of the non-candidate sub-windows of the input image. In order to be defined as a candidate, the detector cascade has four stages, as shown in Figure 3.10, and they are presented next.

Foreground Detection

In this approach, a background model [4] is created and then it is performed a background subtraction in four steps. Firstly, it is calculated the absolute difference, $I_{absDiff}$, between the background model, I_{bg} , and the the image, I .

$$I_{absDiff} = |I_{bg} - I| \quad (3.1)$$

Next, a threshold of 16 pixels is applied to $I_{absDiff}$ to create a binary image, I_{binary} .

$$I_{binary}(x,y) = \begin{cases} 1 & \text{if } I_{absDiff}(x,y) > 16 \\ 0 & \text{otherwise} \end{cases} \quad (3.2)$$

Later, the labeling algorithm proposed by Chang *et al.* [31] is applied to calculate smallest bounding box for the blob originated until this point. Finally, every component from the binary image smaller than the area of the selected bounding box is removed and the final result is created.

Variance Filter

The variance filter method filters out all image sub-windows with variance lower than a pre-defined threshold σ_{min}^2 which is by default half of the initial selected variance value used. The calculation of the variance, image uniformity measure, can be speeded up by recurring to integral images as demonstrated in Equation 3.3 and it is derived from the Equation shown in [28].

$$\sigma^2 = \frac{1}{n} I''(B) - \left[\frac{1}{n} I'(B) \right]^2 \quad (3.3)$$

$I'(B)$ is the sum of pixels and $I''(B)$ is the sum of square of pixels inside a bounding box B for σ_{min}^2 :

$$I'(B) = \sum_{i=1}^n x_i \quad (3.4)$$

$$I''(B) = \sum_{i=1}^n x_i^2 \quad (3.5)$$

Figure 3.11 exhibits the output of the variance filter by marking with a red box the regions of lower variance and with a green box the regions of higher variance.

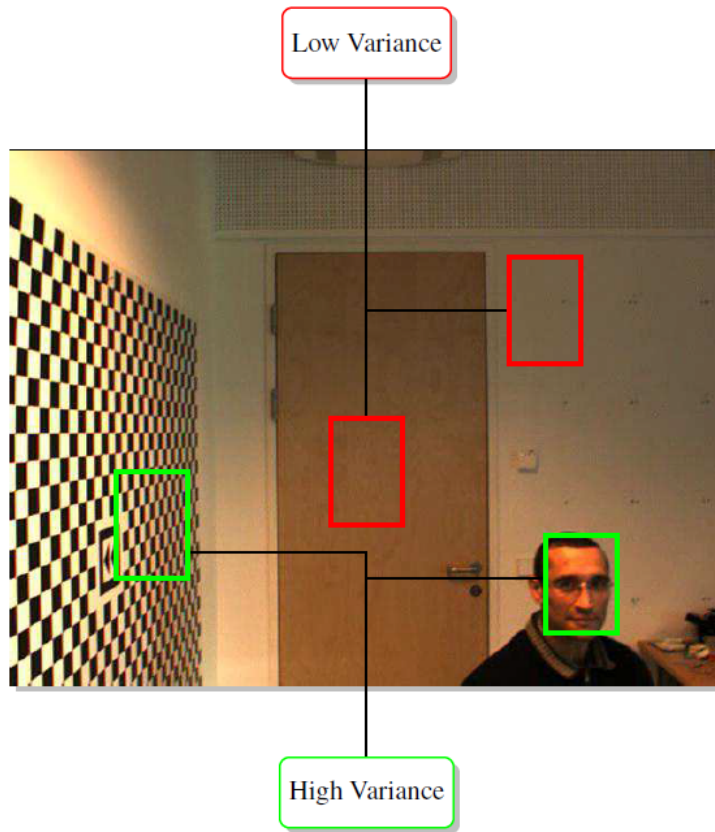


Figure 3.11: Variance filter output [28]

Ensemble Classifier

In the third stage of the detection cascade, a random fern classification [32] is employed. A fern is based on a small set of binary tests which later return the probability that a patch fits to any of the classes learnt during the training step. The classifier decision is made using the number of pair wise pixel intensity comparisons [33]. Each sub-window is then subjected to a rejection criteria if the probability P_{pos} value is lower than the threshold determined.

In Figure 3.12 the feature calculation process is explained by showing the image to be classified and the dots referring to the pair wise pixels in the image. At first, an uniform distribution is used to draw the dots positions and later the intensity between pair of pixels in each box is compared. The comparison is invariant to brightness variations and can be expressed as the Equation 3.6, where $d_{i,1}$ and $d_{i,2}$ are random locations.

$$f_i = \begin{cases} 1 & \text{otherwise} \\ 0 & \text{if } I(d_{i,1}) < I(d_{i,2}) \end{cases} \quad (3.6)$$

Finally, the comparison result is used to retrieve the probability $P(y = 1|F)$ where the value of y refers to the class label of the sub-window, either positive if $y = 1$ or negative if $y = 0$. Figure

3.12 shows the process for a single fern so the calculation of the final confidence value P_{pos} is obtained by averaging the value of multiple ferns.

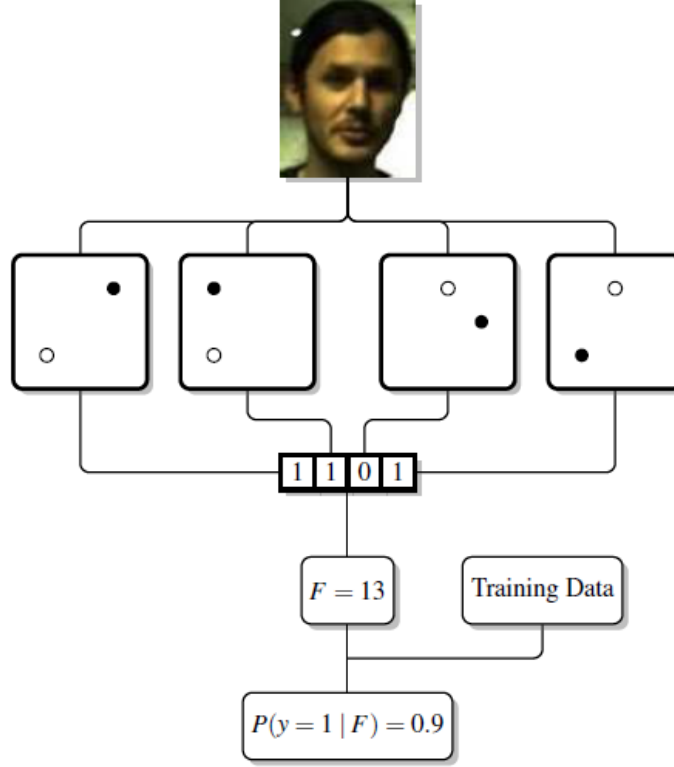


Figure 3.12: Single fern feature calculation [28]

Template Matching

The final stage of the detector cascade is done using template matching and a Nearest Neighbour Classification (1NN). To enable the algorithm's learning, it is necessary to build a list of positive and negative templates. The templates are later translated into patches with the size of 15x15 and by using the Normalized Correlation Coefficient (NCC) [28] it is possible to compare two learnt patches. The NCC Equation (3.7) values vary from 1 to -1 and the closer to 1 the similar the patches are and the opposite for the -1 value.

$$ncc(P_1, P_2) = \frac{1}{n-1} \sum_{x=1}^n \frac{(P_1(x) - \mu_1)(P_2(x) - \mu_2)}{\sigma_1 \sigma_2} \quad (3.7)$$

where μ_i and σ_i are mean and deviations of patches P_i with $i = \{1, 2\}$. The distance of patches can also be calculated using the Equation 3.8 and its results can vary from 0 to 1.

$$d(P_1, P_2) = 1 - \frac{1}{2}(ncc(P_1, P_2) + 1) \quad (3.8)$$

Figure 3.13 shows how the confidence p^+ from the positive class is expressed. This classification is made by combining d^+ and d^- that are, respectively, the distance between an unknown image patch and the closest member of a positive and negative class. This assumption can be translated into the Equation 3.9 and the confidence value achieved can enable a sub-window acceptance if greater than a threshold.

$$p^+ = \frac{d^-}{d^- + d^+} \quad (3.9)$$

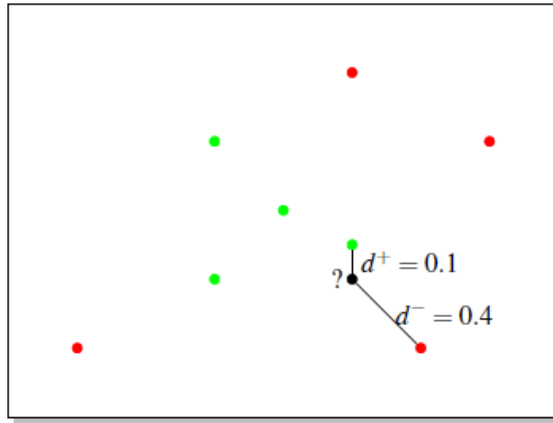


Figure 3.13: Distance classification of unknown patches [28]

In order to finalize the detection process, by the end of the detector cascade there are possible sub-windows candidates for the object. Figure 3.14 demonstrates this issue by showing multiple high confidence sub-windows around the true detection in green. Blaschko [34] determined that the choosing of sub-windows with high confidence is difficult to execute. Instead resorting to non-maximal suppression techniques to combine the result is the most desirable method of implementation.

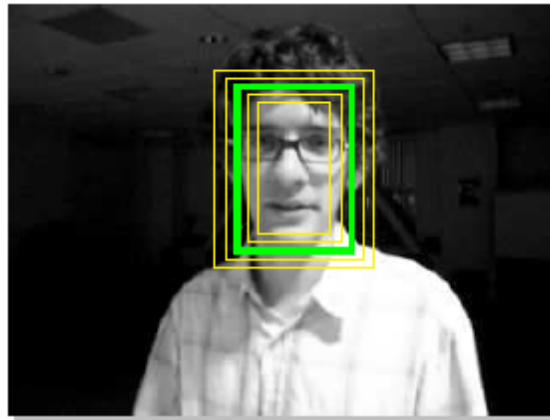


Figure 3.14: High confidence sub-windows (yellow) and true detection (green) [28]

Viola *et al.* [30] describes a method for non-maximal suppression using clusters detections based on spatial overlap. Meaning that for each cluster, the final and single result is a combination and average of the bounding boxes.

Finally, a hierarchical clustering algorithm presented in [35] is used. Its implementation is based on, firstly, the calculation of the pair wise overlap between all bounding boxes with high confidence. Then, when the starting bounding box is chosen the nearest bounding box is checked. The distance is measured and if it is less than a predefined threshold, both bounding boxes are included in a single cluster. If one of these bounding boxes is already in a cluster, they are merged. If the opposite happens, they are included into a different cluster. This procedure is then continued for the bounding boxes that remains.

3.3.1.3 Learning

One of the big issues with this algorithm is how to combine the output of the tracking and detection steps into a final result, like it is shown in Figure 3.8. The learning process requires some criteria over the final result, if those criteria are met than the P/N learning constraints [36] are enforced during the learning step. During this, the ensemble classifier and the template matching method are trained. This section presents how those steps are implemented.

Fusion and Validity

The fusion algorithm is implemented using the following criteria:

- If there is only one detector's result with higher confidence than the tracker's result, then the detector's result is selected. This enables the recursive tracker's re-initialization.
- If a bounding box is estimated by the recursive tracker and is not re-initialized because of the previous point, then the recursive tracker's result is selected.
- If no bounding box is selected, then it is implied that the object is not visible.

Stating the previous criteria, the only points where the final result is considered valid are the ones when the tracker is not re-initialized by the detector. In all the other situations the result is considered invalid unless it follows the next criteria:

- The final result is valid if the previous result was valid and the recursive tracker's confidence output is greater than the threshold σ^+ (positive class) or σ^- (negative class).

P/N Learning

The learning method applied is the semi-supervised learning proposed by Chapelle [37]. This method has labelled examples and unlabelled data which are used as supervisory information for the training data. It distributes in classes the unlabelled data and updates the classifier using the class separation as a training set. In this scenario, the labelled positive example considered is the initial selected bounding box.

Kamal *et al.* [36] introduced the positive and negative constraints which influenced the semi-supervised learning method and enabled the learning by restricting the labelling of the unlabelled data. This method has two types of constraints: P which are used for identification of false negative outputs and its inclusion in the positive training examples; N creates the opposite effect. Basically, the P/N learning classifier evaluates the unlabelled data, removes the cases where the object is classified in disagreement with the structural constraints and adds the right samples to the training set using an iterative method.

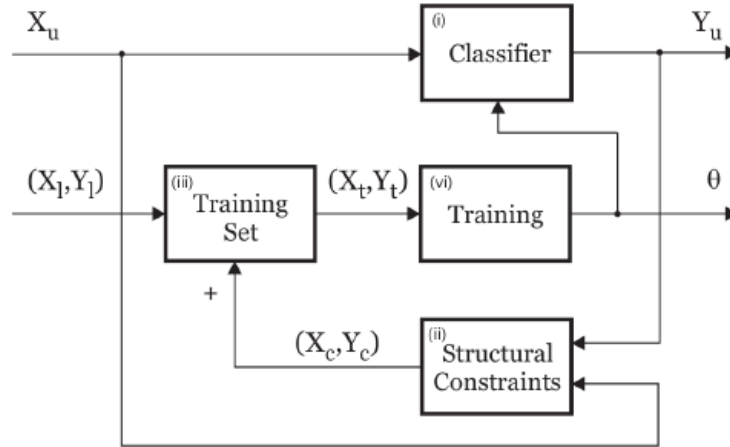


Figure 3.15: P/N Constraints [28]

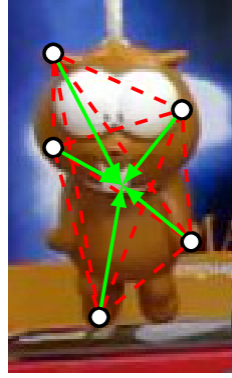
Let x_i and y_i correspond, respectively, to the training examples and its class label. Therefore, while using those variables, the function for the classifier is $f : X \rightarrow Y$ and is applied to the unseen data. Figure 3.15 exhibits the unlabelled data X_u under the action of a classifier that assigns it as Y_u . Under structural constraints, Y_u and X_u , go under some criterion to help identify the misclassified examples X_c and Y_c . Finally, X_c and Y_c are incremented to the training set which will later update the classifier.

The structural constraints used are the ones proposed by Kamal *et al.* [36]. P-constraints affirm that all highly overlapping patches (default: $> 60\%$) with the final result are classified as positive while N-constraints suggest all the non overlapping patches (default: $< 20\%$) with the final result are classified as negative examples.

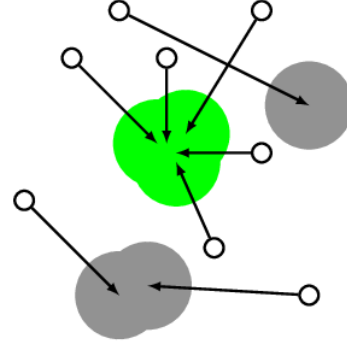
3.3.2 CMT

Nebehay *et al.* [3] proposed an algorithm called Consensus-based Matching and Tracking of Keypoints (CMT). Basically, the main idea and structure behind CMT is to obtain the keypoints from the object of interest by using the BRIEF detector [38]. Later, in each frame it is tried to find the initial keypoints of the object. There are used two methods to find the keypoints in the current frame: the use of the optical flow [13] to estimate the keypoints from the previous frame and the match between keypoints by comparing their descriptors. The problem is that both methods are prone to error the methods referred before are applied to find the consensus between the keypoints

in order to find the center of the object, as shown in Figure 3.16a, and have a more successful tracking algorithm. Figure 3.16b exhibits the keypoints votes that are then clustered enabling the separation of the outlier from the inlier. The remaining keypoints are then used to estimate the new bounding box.



(a) Keypoints consensus and object's center



(b) Clustered keypoint votes

Figure 3.16: CMT tracker principles [3]

Therefore, having explained the principles of the CMT tracker, its framework and the details of each method used are now presented.

3.3.2.1 Matching and Tracking of Keypoints

According to Nebhay *et al.* [3] the method's object model is represented by a set of keypoints in Equation 3.10 where each of them stands for a location $r \in \mathbb{R}^2$ and a descriptor f , this descriptor is binary for computational reasons.

$$O = \{(r_i, f_i)\}_{i=0}^{N^O} \quad (3.10)$$

Given a sequence of images I_1, \dots, I_n and a region of interest b_1 initialize in I_1 , O is initialized when the keypoints in I_1 , which are inside b_1 , are detected and described. Later the keypoints locations are mean-normalized. The object pose can be recovered in the current frame I_t if a set of corresponding keypoints is found. In Equation 3.11 the corresponding keypoints are defined and where a and m respectively refer to the keypoint location in I_t and is the corresponding keypoint index in O .

$$K_t = \{(a_i, m_i)\}_{i=1}^{N^{K_t}} \quad (3.11)$$

K_t can be found by the use of two complementary methods based on matching and tracking of keypoints. Candidate keypoints are detected as described in Equation 3.12, and for each it is

computed the Hamming distance of its descriptors to the I_1 descriptors in Equation 3.13.

$$P = \{(a_i, f_i)\}_{i=1}^{N^P} \quad (3.12)$$

$$d(f^1, f^2) = \sum_{i=1}^d \text{XOR}(f_i^1, f_i^2) \quad (3.13)$$

The candidate keypoints are matched in P to keypoints in I_1 by applying the second nearest neighbour distance criterion [24] on the descriptors distance d . Later, the matched keypoints set M is composed by the keypoints in P that match O . In the opposite sense, keypoints that match the background keypoints are removed from M .

The tracking process is done by computing the displacement of keypoints in consecutive frames by applying the method of sparse optical flow [13]. Later, the set of tracked keypoints T is done by removing the wrong correspondences by using the forward-backward error measure [29].

In the end, T and M are fused into a set K' of size $N^{K'}$, promoting the discardment of tracked keypoints if there is a matched keypoint correlated with the same model keypoint. Accordingly to Nebehay [3] because matched keypoints do not depend on recursive estimation they are more robust. Nevertheless, K' can still contain outliers because there is still some uncertainty in the matching and tracking processes.

3.3.2.2 Voting

The location of the object of interest is defined by casting votes h from the candidates keypoints K' for the object center, culminating in a set of votes V as shown in Figure 3.16a.

$$V = \{h(a_i, m_i)\}_{i=1}^{N^{K'}} \quad (3.14)$$

$$(3.15)$$

If the vote simply suffers translational changes then the vote is represented as presented in Equation 3.17. The appearance of scale changes affects the voting cast. In order to overcome this problem the votes are scaled by a factor s as shown in Equation 3.17.

$$h^T(a, m) = a - r_m \quad (3.16)$$

$$h^S(a, m) = a - s \cdot r_m \quad (3.17)$$

The next step is to compute statistics over pairwise geometric properties between the estimated pairs, expressed by $a^{i,j}$ and $r^{i,j}$. Being $a^{i,j}$ the Euclidean distance between a_i and a_j in K' , while $r^{i,j}$ the Euclidean distance between r_{mi} and r_{mj} in O . To perform such calculations it is adopted existing heuristics for estimating s . Kalal *et al.* [29] proposed the estimation of the scale s as in

Equation 3.18.

$$s = \text{med} \left(\left\{ \frac{\|a^{i,j}\|}{\|r^{i,j}\|}, i \neq j \right\} \right) \quad (3.18)$$

The appearance of an in-plane rotation in the object creates the necessity of updating the voting system accordingly to the rotation appeared. Therefore updating Equation 3.17 with a 2D rotation matrix R in order to α .

$$h^R(a, m) = a - s \cdot R r_m \quad (3.19)$$

$$R = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} \quad (3.20)$$

The rotation α of the object is possible to robustly estimate from the distribution of the pairwise angular changes and later computing its median like in Equation 3.21, proposed by Nebehay *et al.* [3].

$$\alpha = \text{med} \left(\left\{ \text{atan2}(a_y^{i,j} - a_x^{i,j}) - \text{atan2}(r_y^{i,j} - r_x^{i,j}), i \neq j \right\} \right) \quad (3.21)$$

3.3.2.3 Consensus

In this section, the outlier keypoints and their respective votes are identified and removed by searching for consensus in the voting behaviour in order to find the object center μ , Figure 3.16b shows examples of that behaviour. To achieve this result, a hierarchical agglomerative clustering [39] is applied on V , based on the Euclidean distance as a dissimilarity measure. Using this clustering technique it is considered that the subset with the most number of elements is the consensus cluster V^c and K^t the set that voted for V^c . One of the advantages of this method is the nonexistence assumptions about the object planarity because the keypoints have a degree of flexibility δ regarding the keypoints' drift.

In case V^c has fewer elements than $\theta \cdot |O|$ elements, the object is determined as non visible. In the opposite case, the votes used for the consensus cluster are turned into an estimate for the object center as shown in Equation 3.22, where $n = |V^c|$.

$$\mu = \frac{1}{n} \sum_{i=1}^n V_i^c \quad (3.22)$$

The final object bounding box can be achieved with the help of μ , s and α . The corners of the bounding box can be retrieved using the Equation 3.23.

$$c'_i = \mu + s \cdot R c_i \quad (3.23)$$

3.3.3 Software Implementation

In this section it will be explained how the system is implemented using the described tracking algorithms in the architecture referred in section 3.1.

The algorithms presented in sections 3.3.1 and 3.3.2 were developed under OpenCV API and using specific C++ libraries developed by Nebehay [40, 41].

The implementation was based in two different processes. The first one based on the user interface to select the object of interest by determining the ROI and the second where the selection of the object is made automatically.

The first method involves using one of the two algorithms for the processing step recurring to the functions supplied by Nebehay in his C++ libraries while doing the rest of the steps using the C++ OpenCV API. The implementation for the tracker using TLD or CMT relies respectively in the algorithms 1 and 2.

Algorithm 1 TLD	Algorithm 2 CMT
Input: $I_1 \dots I_n$	Input: $I_1 \dots I_n$
1: while $t = 1 \dots n$ do	1: while $t = 1 \dots n$ do
2: if aux is <i>true</i> then	2: if aux is <i>true</i> then
3: $R_t \leftarrow \text{track}(I_{t-1}, I_t, B_{t-1})$	3: $P \leftarrow \text{detect}(I_t)$
4: $D_t \leftarrow \text{detect}(I_t)$	4: $M \leftarrow \text{match}(P, K_O)$
5: $B_t \leftarrow \text{fuse}(R_t, D_t)$	5: $T \leftarrow \text{track}(K_{t-1}, I_{t-1}, I_t)$
6: if valid(B_t) then	6: $K' \leftarrow T \cup M$
7: learn(I_t, B_t)	7: $(s, \alpha) \leftarrow \text{estimate}(K', K_O)$
8: end if	8: $V \leftarrow \text{vote}(K', K_O, s, \alpha)$
9: print(B_t)	9: $V^c \leftarrow \text{consensus}(V)$
10: end if	10: $K_t \leftarrow \text{vote}^{-1}(V^c)$
11: if select(B_t) and aux is <i>false</i> then	11: if $ V^c \geq \theta \cdot K_O $ then
12: learn(I_t, B_t)	12: $B_n \leftarrow \text{boundingbox}(B_t, \mu, s, \alpha)$
13: aux \leftarrow <i>true</i>	13: print(B_n)
14: end if	14: end if
15: end while	15: end if
	16: if select(B_t) and aux is <i>false</i> then
	17: $K_O \leftarrow \text{detect}(I_t, B_t)$
	18: aux \leftarrow <i>true</i>
	19: end if
	20: end while

This implementation was firstly done in an UNIX environment and then ported to the Android platform as explained in section 3.2.

The second method relies in an automatic process to detect objects, in this case people. Therefore, in the algorithm there is a change in the process. Instead of manually selecting the region of interest, the ROI is automatically given by a detector. The detector is the one presented in section 2.1.3.2 called Histogram of Oriented Gradients [23]. The SVM classifier was trained using the INRIA dataset and provides a default people detector when using HOG algorithm. In conclusion, it is possible to transform the algorithms 1 and 2 by modifying the part that represents the user interface and by adding the HOG algorithm to detect the bounding box for the region of interest, creating an automated process desired for the surveillance application.

3.4 Discussion

In this chapter the system overview was presented, as well as the software platforms and the algorithms used for the tracking purpose.

The system overview explains how the system was structured in order to facilitate the wanted outcome of this dissertation. The equipment and software used are explained and reviewed to get a better understanding on how to use it correctly for this purpose. The software issue is one of the biggest challenges in this system overview because it is essential when the material to be developed is a software application. In that sense, it was studied how the portability between an UNIX and Android platform could be done using the OpenCV library. The use of the Android NDK is the perfect strategy for the portability wanted and even giving better performance results when comparing to the Java API use.

The tracking algorithm issue was resolved by using TLD and CMT. Both of them are able to track unknown objects from the start and are able to behave in dynamic environments. The system being mobile presents an instability when referring to the video capture and it is necessary to have algorithms capable of tracking objects in those circumstances. It is also necessary to add to those algorithms an automatic people detector. Since it is a mobile surveillance system, the HOG algorithm was combined instead of depending on the user interface developed to choose the object of interest.

Chapter 4

Evaluation

In this chapter, the TLD and CMT algorithms are evaluated empirically on some sequences provided by the dataset of the Bonn Benchmark on Tracking (BoBoT) ¹. The method used to evaluate the algorithm performance quantitatively relies in the employment of the standard metrics recall and precision. Qualitatively the algorithm's output will be discussed of the dataset sequences exhibit. In the end, the differences between the performance in the Unix and Android environment will be discussed .

All experiments were made under Intel® Core™ i7-3770S processor @ 3.10GHz.

4.1 Evaluation Method

In order to evaluate the algorithms performance it is necessary to compare the algorithm output against the ground truth values provided by the BoBoT dataset. This comparison is possible by measuring the overlap between the two bounding boxes. Therefore, the equation 4.1 is used from the PASCAL challenge [42] to measure the overlap.

$$overlap = \frac{B_1 \cap B_2}{B_1 \cup B_2} = \frac{I}{B_1 + B_2 - I} \quad (4.1)$$

where B_1 and B_2 are respectively the areas of the two bounding boxes and I the area of the intersection. Figure 4.1 illustrates this measurement.

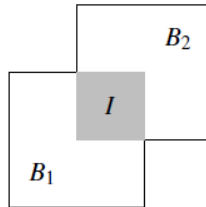


Figure 4.1: Overlap measurement [28]

¹Bonn Benchmark on Tracking: <http://www.iai.uni-bonn.de/kleind/tracking/>

Matthews *et al.* [43] demonstrate that this overlap measure penalizes changes in direction and scale. Therefore, the evaluation method is based in five situations between the algorithm output and the ground truth as shown in Figure 4.2.

The result of this method can be considered as:

- **True Positive (TP)**: Overlap satisfies the condition of the threshold ω .
- **False Negative (FN)**: The ground truth exist but there is no algorithm output.
- **False Positive (FP)**: The algorithm output occurs but the ground truth does not exist.
- **FN and FP**: Overlap does not satisfies the condition of the threshold ω
- **True Negative (TN)**: Neither ground truth or algorithmic output exist.

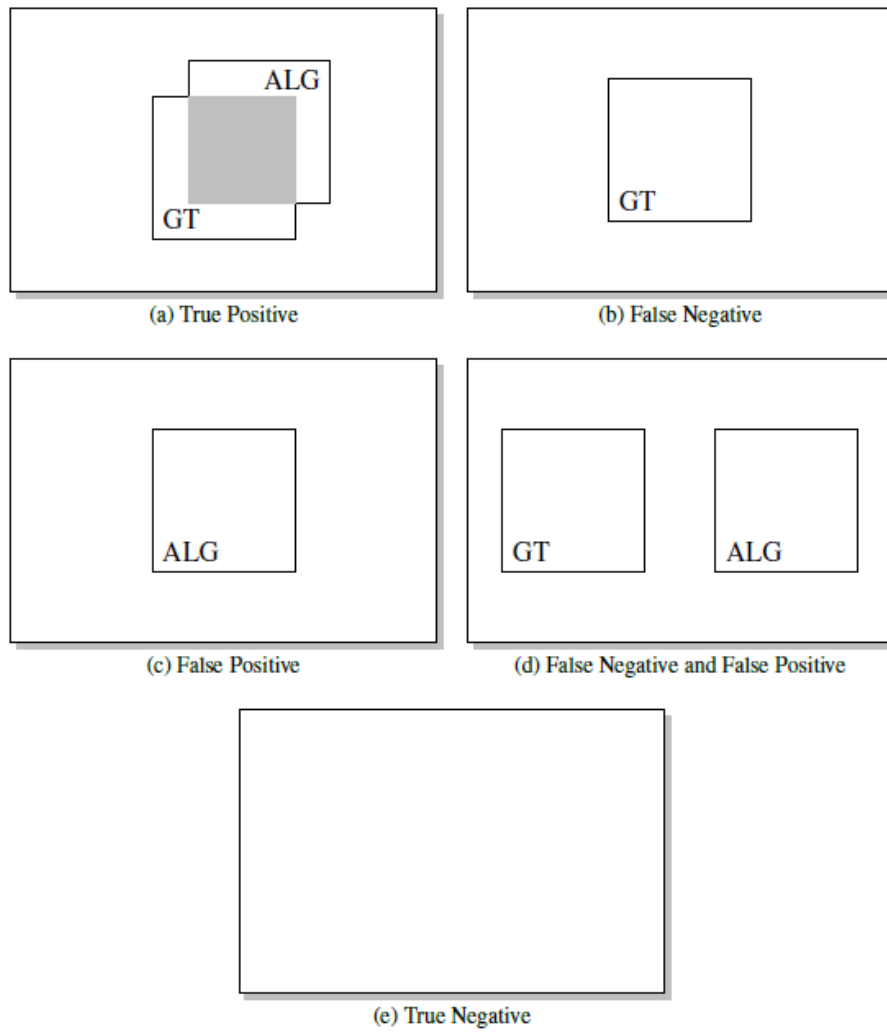


Figure 4.2: Overlap comparison cases [28]

The performance of the algorithm is normally expressed in terms of recall and precision [44]. These parameters can be defined by using the number of occurrences of TP, TN, FN and FP in the video evaluation. Consequently, recall measures the fraction of positive examples that are classified correctly and is defined by the equation 4.3 while precision measures the fraction of examples labelled positive that are actually positive and is defined by the equation 4.3.



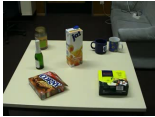
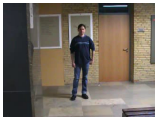



$$recall = \frac{TP}{TP + FN} \quad (4.2)$$

$$precision = \frac{TP}{TP + FP} \quad (4.3)$$

4.2 Dataset

In this section, the sequences used for evaluation are characterized. The final result of this dissertation is an Android application and therefore it was necessary to have a dataset where the video sequences were dynamic and had different situations and objects of interest. These specifications lead to the sequences from BoBoT [1]. The sequences used are described in Table 4.1.

Table 4.1: Description of BoBot Sequences

Seq.	Frame	1st Frame	Target Object	Attributes
A	602		Ball	Moving cam / Moving target / Rotation / Fast direction change
B	629		Cup	Moving cam / Moving target / Background changes / Scale changes
C	404		Juice box	Moving cam / Fast direction changes / Scale changes
E	305		Person	Moving cam / Partial occlusion
G	716		Cube	Moving cam / Viewpoint changes
I	1017		Person	Moving cam / Moving target / Rotation / Similar distractors / Full occlusion / Outdoor
K	1020		Cup	Moving cam / Viewpoint changes / Similar distractors / Scale changes

All sequences provided by BoBoT also have available the ground truth data and have a default size of 320x240. The availability of a ground truth allows the use of the evaluation method proposed in Section 4.1. The ground truth annotations have the target's position relative to the frame size and the size defined as the tightest-fitting rectangle surrounding the object.

4.3 Results

In this section, the qualitative and quantitative results obtained using the algorithms proposed in Section 3.3 will be discussed under the dataset presented in this chapter. The qualitative evaluation rests upon the visualisation of the algorithmic output in the video sequences. On the other hand, the quantitative evaluation is based in the analysis of the recall and precision values obtained while running the algorithms.

4.3.1 Qualitative Results

4.3.1.1 Dataset Results

The qualitative evaluation is made for the TLD and CMT results over the sequences presented in Table 4.1.

It is possible to verify in Figure 4.3 that the TLD algorithm in the sequences A, I and K does not behave as well as in the other sequences. In case A, the reason for the nonexistent output is because of the fast movement of the object. In this scenario the learning step fails to update the detector properly, therefore not enabling the estimation of the new bounding box. Only when one of the learnt templates appear again in the frame, the detector is able to re-initialize the tracker. In case I and K, the condition affecting the performance of the TLD algorithm is the appearance of similar distractors present in the background. In sequence I, as the object of interest is crossed over and partially occluded by a similar object, the learning step estimates that the other object is the correct template and updates the detector with the wrong templates, leading to a wrong estimation of the bounding box and learnt patches. The same problem occurs with sequence K, but this time it is due to the viewpoint changes and not because of the crossover between objects.

The CMT algorithm shows an inability for the tracking process over the sequence B as shown in Figure 4.4. This inefficiency is translated by the adaptive model which enables the estimation of the keypoints in the current frame using optical flow. The background being rather cluttered allows the estimation of keypoints to be error prone, defining background keypoints as candidates and translating them as input for the estimation of the new bounding box used for the output function. As it is possible to observe in sequence B, along the frames the bounding box represented in blue starts to gradually increase, including some keypoints of the background instead of solely having as reference the object of interest keypoints.

The comparison between TLD and CMT differs completely when it comes to the failure points presented. Although both algorithms present good results in most of the BoBoT scenarios tested.

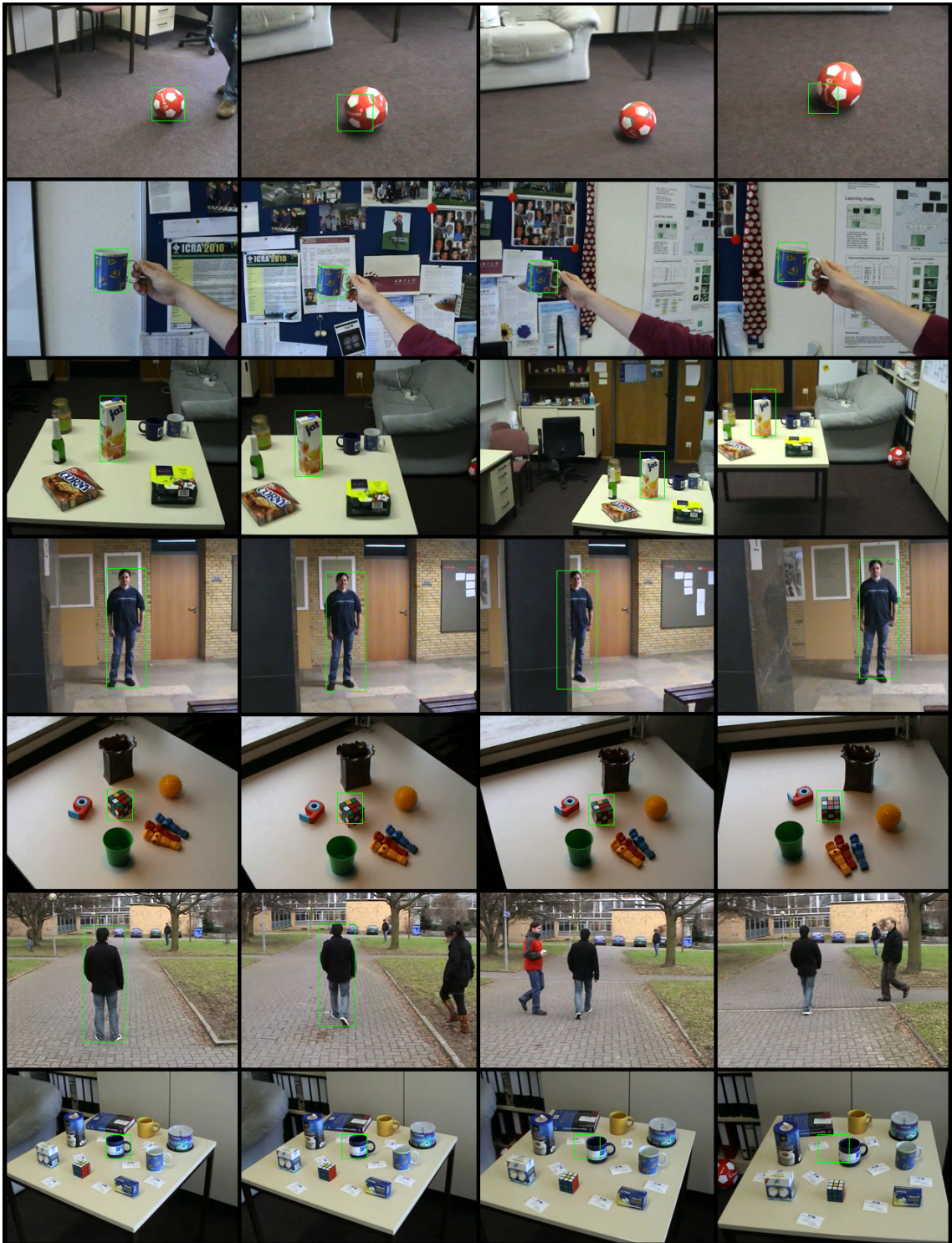


Figure 4.3: TLD algorithm output (green region) over the BoBot video sequences of A, B, C, E, G, I and K (top to bottom) from frames 1, 100, 200 and 300 (left to right)

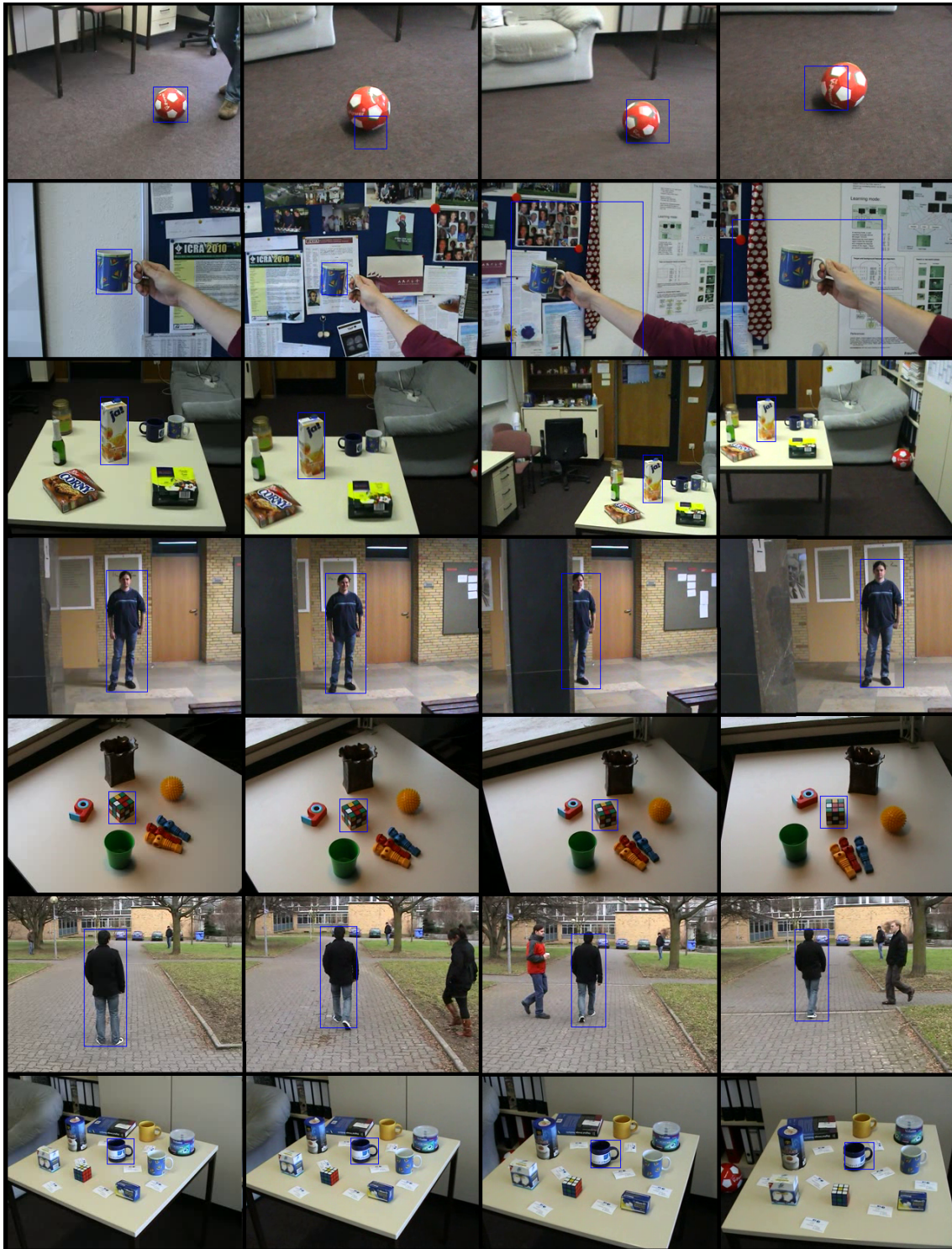


Figure 4.4: CMT algorithm output (blue region) over the BoBot video sequences of A, B, C, E, G, I and K (top to bottom) from frames 1, 100, 200 and 300 (left to right)

4.3.1.2 Android Application Results

In this section, several sequences of images resulted from the Android application will be exhibited and analyzed.

The sequences presented have different specifications and attributes, which can be seen in Table 4.2. These sequences were chosen to determine how the application behave in different environments and to determine the accuracy of the algorithms implemented in the Android environment. The final system of the MOTrack application use the CMT for tracking while the detection is done by using HOG. This implementation was chosen because, as discussed and seen in section 4.3.1.1, CMT has shown better results for tracking people.

Table 4.2: Image sequences obtained from MOTrack application

Sequence	Algorithm	Target Object	Attributes
A1 / A2	TLD / CMT	Can	Mov. Cam / Simple Back.
B1 / B2	TLD / CMT	Charger	Mov. Cam / Cluttered Back.
C1 / C2	TLD / CMT	Can	Mov. Cam / Mov. Obj. / Cluttered Back.
D1 / D2	TLD / CMT	Book	Mov. Cam / Mov. Obj. / Cluttered Back.
E F G H I	HOG+CMT	Person	Mov. Cam / Mov. Obj. / Cluttered Back.

The sequences from A to D are done using the manual selection of the object, while the others are done using the the people detector.

Sequence A

In sequence A1 and A2, they use, respectively, the algorithms TLD and CMT. Figure 4.5 shows the output of the MOTrack application using TLD and the object is correctly tracked during the sequence. One of the problems with the output comes from the similarity between the background and the object of interest. The primary colour of both, background and object, is white causing the tracking process of the TLD algorithm to start considering some keypoints from the background as candidate keypoints. This situation causes the algorithm to start considering the background as being part of the object of interest. Nevertheless, it is able to track the object during its full rotation movement.



Figure 4.5: Sequence A1

Figure 4.6 shows the output of the MOTrack application using CMT and the object is not tracked during the all sequence. This situation occurs because CMT uses keypoint matching to

track the object. The initial region of interest contains the keypoints of the frontal surface of the object, so when the camera is facing the object's back it cannot track because its surface is different from the initial surface selected. That occurrence is exhibited in the last image of the Figure 4.6. Although, when the camera is facing towards the frontal side of the object, the tracking is reinitialized successfully. The situation where the background starts to get tracked, as shown in sequence A1, does not happen.



Figure 4.6: Sequence A2

Sequence B

It is possible to observe in Figure 4.7 and 4.8 that both algorithm's output work accordingly to expected. In the images of both sequences it is possible to see, along the video frames, that the object continues being tracked even though the background is completely cluttered. One of the reasons for this to happen is the existence of an abundant amount of keypoints in the region of interest selected, allowing the tracking algorithms to perform well in terms of scale and rotation.

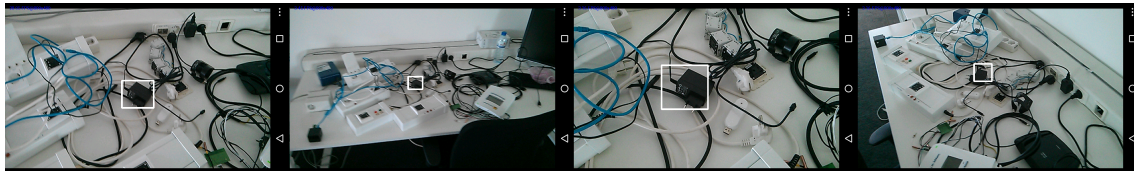


Figure 4.7: Sequence B1



Figure 4.8: Sequence B2

Sequence C

The following sequences, exhibit images with moving camera and object as attributes. The results using the TLD and CMT can be seen, respectively, in Figures 4.9 and 4.10. The TLD output, as it is possible to see in Figure 4.9, demonstrates an inability to track the object when it passes through background zones that are similar to the object of interest. Therefore, from the

second to the third image it is possible to observe that after crossing over a white background zone, the tracking process fails to continue. This is due to the tracking algorithm starting considering the background as object of interest and, therefore, training the classifier with the incorrect dataset, which precludes the reinitialization of the tracking stage of the TLD algorithm.

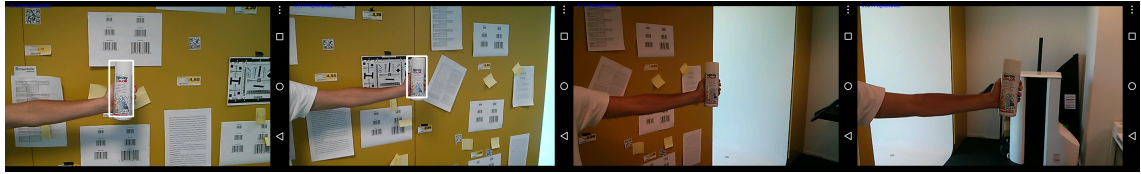


Figure 4.9: Sequence C1

Concerning the CMT algorithm, as Figure 4.10 demonstrates, the problem that occurs in the TLD algorithms also occurs when the background is completely white. This is due to the similarity of the object of interest keypoints with the background keypoints. This similarity promotes an error in the matching stage of the CMT algorithm, considering as candidate keypoints the ones from the background which results in an inefficiency when it comes to the algorithm's output.

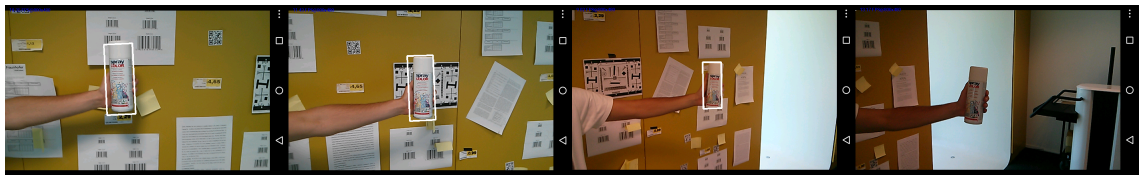


Figure 4.10: Sequence C2

Sequence D

Sequence D presents similarities with sequence C but has one major different attribute. This difference comes from the object of interest having a colour not similar with the ones in the background. This attribute should improve the results of the algorithms comparing to the results from sequence C.

Figure 4.11 shows the TLD algorithm working properly all over the sequence. These results exist because the features of the object of interest are completely different from the ones of the background, providing a correct tracking stage during all the process.

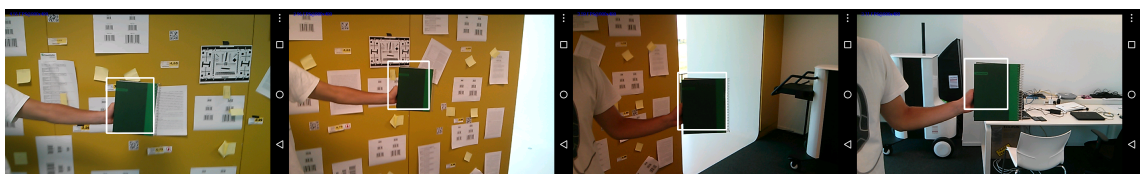


Figure 4.11: Sequence D1

Figure 4.12 represents the CMT algorithm's output and shows that, comparing to the TLD algorithm, it does not have the same results. The problem comes from the flat surface of the object of interest. One of the main stages of this algorithm is the matching of keypoints, which uses the BRISK as detector and descriptor. The surface of the object being flat, the BRISK detector is unable to detect a large amount of keypoints because there are no contours. Therefore, with the small amount of keypoints detected is harder for the algorithm to determine the region of interest. So, in the third image of the Figure 4.12, as there is a sudden change of background, the algorithm stops tracking the keypoints selected because it cannot use the matching stage to select the candidate keypoints. This occurred because the initial region of interest detected a small number of keypoints, enabling the initial tracking process to be done using the optical flow method other than the matching method, and when the matching method was necessary to determine the candidate keypoints, the algorithm could not solve its problem because of the initial detection of the keypoints.

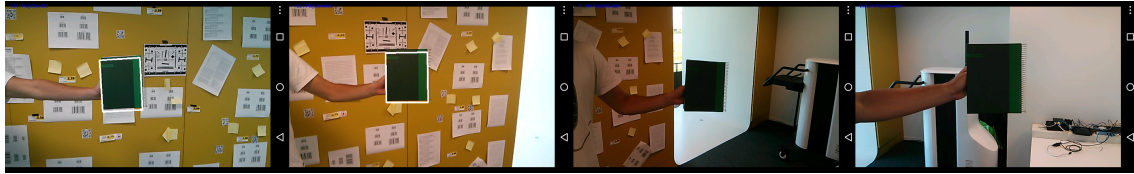


Figure 4.12: Sequence D2

From this part of this section, the sequences to be shown are a result of the algorithm's output using the HOG for the people detector and the CMT for the tracking algorithm. These sequences have as primary object of interest a single person in different scenarios and backgrounds.

Sequence E

Figure 4.13 shows the recorded sequence and the algorithm's output obtained. In the first image of the sequence, it is possible to see the result of the HOG algorithm correctly identifying the person displayed in the scenario. In the other images, it is possible to observe that the CMT algorithm is able to track the object of interest obtained using the people detector. The person in this sequence presents a walking route where rotation and scale changes are present. Even with these changes, the algorithm is able to detect and track the person along the video frames.

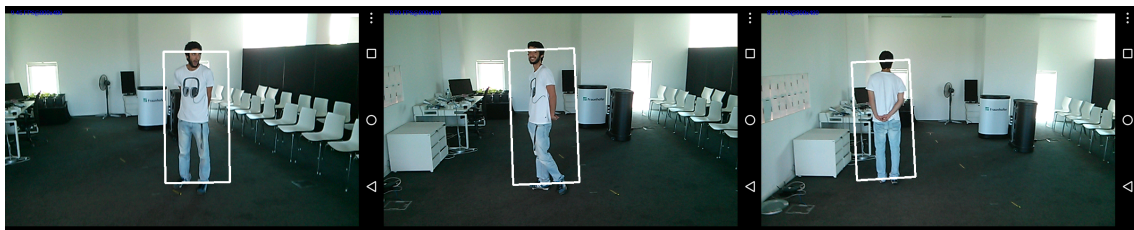


Figure 4.13: Sequence E

Sequence F

Sequence F is similar to sequence E, they have the same background and the movement made by the person is identical, although the results are not the same. Figure 4.14 represents sequence F and in the last image is possible to visualize that the tracking algorithm is not tracking correctly. This phenomenon is due to the region of interest obtained using the HOG algorithm. In sequence F, by contrast with sequence E, the region of interest obtained is bigger and therefore contains more background elements and its keypoints. The CMT algorithm relies on keypoints matching and optical flow tracking to determine which are the candidate keypoints. In the third image of the sequence, the algorithm's output is based on the result of the matching stage. The region of interest is the same as the one of the first image which resulted from the output of the people detector. This means that after some point, the tracking algorithm considers the keypoints from that area as the candidates because the initial region obtained has a large amount of keypoints that refer to the background. Therefore, the algorithm determines the region that contains those keypoints as the output, although it is incorrect and does not correctly follow the person.



Figure 4.14: Sequence F

Sequence G

Figure 4.15 represents sequence G and it is possible to analyze that the algorithm works correctly over all the sequence. The sequence exhibits changes in the background while the person is moving in terms of rotation and luminosity. These changes do not affect the outcome of the algorithm allowing it to perform correctly over the sequence. Sometimes the bounding box do not represent entirely the object of interest, containing parts of the background, like in the second image but it still is able to follow the person correctly. Depending on the precision that the user wants for this application, this sequence is an example that even though the precision is not the best one, the algorithm is able to track the person along the frames with some accuracy.

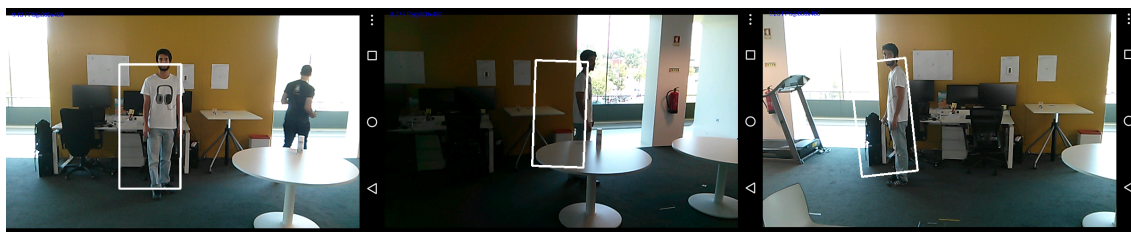


Figure 4.15: Sequence G

Sequence H

The sequence H, as shown in Figure 4.16, results in the tracking of a person in an hallway. From the beginning to the end of the sequence, the user and the object of interest move into each other and the application is able to continuously track the person. The result of the people detector, as seen in the first image, covers the whole person's body and increasing the proximity to the camera, it starts to cover less of the body in each frame. In the last image presented in Figure 4.16 the bounding box is shown merely covering the upper body part of the person.

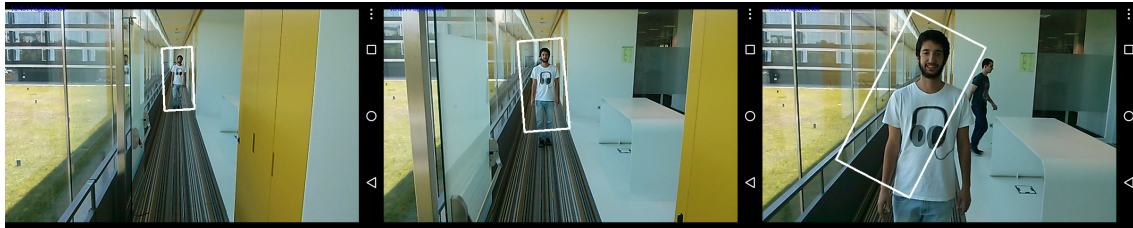


Figure 4.16: Sequence H

Sequence I

The last sequence of the dataset acquired using the MOTrack application is shown in Figure 4.17. The scenario and background is similar to the one presented in sequence H, but it wants to track a person's back and instead of the user and person run into each other, the user follows the person through an hallway. The first image of the sequence shows the result of the people detector algorithm, one of the problems with this case is the non total identification of the person. Instead the algorithm just detected the upper body part of the person. Although this partial detection, it provides a better region of interest for the tracking algorithm to compute. This region of interest is smaller and just contains elements of the object of interest, allowing the tracking algorithm to track more vigorously the person, because it does not contain background keypoints that can mislead the algorithm.



Figure 4.17: Sequence I

In the end of the sequence's analysis, it is possible to determine that the application itself runs properly. In some situations, it is unable to track efficiently, but in most cases it does. One of the problems comes from the HOG algorithm for the people detector. Most of the times, its use translates in obtaining a region of interest way larger than the object of interest, which makes

the detection of keypoints from the background as the ones from the object of interest. That phenomenon lead to the tracking algorithm to have worst results than if the region of interest corresponded solely to the object of interest.

4.3.2 Quantitative Results

In Section 4.1 the relation between the parameter ω and the algorithm output was presented. The parameter ω is responsible for the percentage of overlap that is needed between the result of the algorithm and the ground truth. The increase of ω implies a decrease of both the recall and precision. These requirements can differ depending on what the application is intended for; the more accurate, the bigger the ω .

In this section, how the recall and precision change by modifying ω on both algorithms is analysed. The sequences shown in Table 4.1 are used and three different ω values are defined (1/3, 1/2 and 2/3) for running the tests.

TLD and CMT recall and precision values obtained using the three different values for ω are respectively shown in Table 4.3 and Table 4.4.

Table 4.3: TLD performance with three different values of ω . The values exhibited are respectively the precision and the recall values.

	$\omega = 2/3$	$\omega = 1/2$	$\omega = 1/3$
A	0.38/0.50	0.47/0.63	0.53/0.71
B	0.67/0.67	0.75/0.75	1.00/1.00
C	1.00/1.00	1.00/1.00	1.00/1.00
E	0.91/0.91	0.91/0.91	0.91/0.91
G	0.69/0.69	0.83/0.83	0.97/0.97
I	0.14/0.55	0.14/0.56	0.15/0.58
K	0.10/0.10	0.27/0.27	0.57/0.57

Table 4.4: CMT performance with three different values of ω . The values exhibited are respectively the precision and the recall values.

	$\omega = 2/3$	$\omega = 1/2$	$\omega = 1/3$
A	0.48/0.49	0.75/0.76	0.91/0.92
B	0.21/0.26	0.21/0.26	0.22/0.27
C	1.00/1.00	1.00/1.00	1.00/1.00
E	0.92/0.92	0.92/0.92	0.92/0.92
G	0.64/0.70	0.64/0.70	0.64/0.70
I	0.72/0.76	0.74/0.78	0.75/0.79
K	0.72/0.72	0.77/0.77	0.81/0.81

These results demonstrate the same output as discussed in Section 4.3.1.1 and show an improvement of the algorithmic output when a more relaxed ω is applied. The results of recall and precision for TLD are the lowest for the sequences A, I and K and for CMT for the sequence B. These numbers confirm the analysis performed in the qualitative evaluation. In these tests, frames per second that each algorithm produced were also calculated. The algorithms' processing speed, achieved while performing these tests using the BoBoT dataset, can be seen in Table 4.5. Therefore, CMT processes information faster, in average performs better in more type of scenarios and the variation of ω does not change the results as significantly as in the TLD algorithm, defining a more stable algorithm independently of the requirements applied.

Table 4.5: Algorithms processing speed in frames per second (fps) using Desktop application

TLD	CMT	HOG
15 - 20	25 - 30	10 - 13

These quantitative results exhibited in Tables 4.3 and 4.4 are transposable to the MOTrack application developed. The main difference is the processing speed while running these algorithms and its mainly because of the less powerful hardware that smartphones have comparing to PCs. All experiments in Android were made using the smartphone model LG Nexus 4 E960 with the processor Qualcomm APQ8064 Snapdragon S4 Pro @ 1.5GHz and running Android 5 version. The use of this model translates in the processing speed exhibit in Table 4.6, this was retrieved by changing the image resolution in the MOTrack application and measuring the fps in the different scenarios.

Table 4.6: Algorithms processing speed in frames per second (fps) using MOTrack application

TLD	CMT	HOG	Size
4 - 5	9 - 11	1	800x640
4 - 5	9 - 11	2	640x480
4 - 5	10 - 12	8	320x240

The problem with the automated process is that the people detector is very costly, providing really low FPS for the process. This feature detection functions poorly if people in the scene are moving fast that translates into a wrong initial bounding box, because the frame rate cannot follow up with the people movement. The algorithm used for the automated process was the CMT because from the analysis done it was the one with better results in tracking people scenarios and because of its processing speed. Table 4.6 also demonstrates that the decrease of image resolution affects the HOG algorithm's performance. This is due to the HOG sliding window searching method and its default detection window's size is 8x8. Therefore, the smaller the resolution, the faster the processing speed of the HOG algorithm because the total scan of the frame is made in less steps. In the other hand, the decrease of resolution leads to a deficit in terms of performance in detecting people, creating an issue between the improvement of speed or performance.

4.4 Discussion

The results obtained in this chapter are the key references for the algorithms performance and their evaluation. The method using the overlap ω is rather accurate to evaluate the type of software developed in this dissertation. The recall and precision parameters are fundamental to evaluate the performance of the algorithms in different sequences. The BoBoT sequence was used because it demonstrated the requirements needed for the tests to be performed. The main requirement was that it would contain videos of a freely moving camera and all of the sequences have that specification. Also, it has a lot of different scenarios with different parameters where the algorithms could be tested. In terms of the results of the MOTrack application, the sequences retrieved using the application give us a detailed analyze about the type of occurrences in different scenarios. Most of the scenarios led to positive conclusions, although there is still plenty of room for improvement.

Chapter 5

Conclusions and Future Work

In this project an approach for real-time video tracking objects based on an Android platform for surveillance issues was presented. Android smartphones allow a cheap, flexible, mobile and reliable platform to acquire images from daily scenarios and a great portability for the user to use. However, due to dynamic video sequences, the usual surveillance techniques cannot be performed because normally these systems are treated in static environments. Therefore, the need of having algorithms capable of tracking unknown objects in a dynamic environment is necessary. Also, to enable the use of a smartphone for this purpose, the algorithms should not display a high computational cost. Nowadays, there is no commercial solution available in the market for the project developed in this dissertation.

In this work, a framework was applied to obtain the best capable results from the environments used. This project followed different stages are:

- The definition of the algorithm implemented was done by studying the state of art of tracking algorithms available nowadays. During this study, several algorithms were reviewed and at the end the TLD (Tracking Learning Detection) and CMT (Consensus-based Matching and Tracking of Keypoints) algorithms were chosen to be implemented. The reason behind this decision was the ability of these algorithms to track unknown objects in dynamic scenarios. TLD relies on online learning of the templates retrieved from the object of interest to update the detector and improve the tracking process, whereas the CMT is based in the detection of the object's keypoints and finding the consensus between the keypoints from the object and the image. Both algorithms provide outputs considered relevant for the purpose of this dissertation.
- The implementation of these algorithms in Linux are done using the OpenCV C++ API. The OpenCV library has a high number of functions related to computer vision, so the implementation of computer vision software is facilitated by its use. One of the problems with OpenCV is its non-optimization in some functions enabling the creation of some applications with high computational cost. This implementation detail is critical for this project.

The final platform is an Android smartphone so it is necessary to have algorithms with low computational cost to allow a good frame rate in the application's final output.

- The final platform is an Android application and therefore the software developed in Linux is ported to Android using the NDK tools. Having as resource the Android OpenCV library it was just necessary to create a C++ class to be executed when the Java Native Interface is called from the main Java program creating the GUI of the application.

These different stages lead to the final product of this project: an Android real-time tracking application.

In order to evaluate this application, a dataset was used and an evaluation method based in the overlap between the algorithmic output and the dataset ground truth was performed. The overlap was then classified as 5 different scenarios that were used to calculate the recall and precision of the algorithm. Using these parameters it was possible to evaluate quantitatively the algorithmic output in different scenarios. The results achieved were satisfying and translated in which of the algorithms was performing the best.

Considering the final results presented it is possible to determined that the goals proposed for this dissertation were achieved, despite the improvements that can still be done in some aspects of the project.

5.1 Future Work

During the ongoing dissertation there were some aspects in the project considered problematic for the tracking application. These problematic aspects were found when implementing some features of the application. However, these conditions can be solved by upgrading the system and improving it. Some of the problematic features to be solved in a future work should include:

- Both algorithms should be extended to track multiple objects instead of only one.
- The people detector algorithm should be optimized in order to require less computational cost.

References

- [1] Dominik A., Dirk S., Simone F., and Armin B. Adaptive real-time video-tracking for arbitrary objects. In *IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 772–777, Oct 2010.
- [2] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1409–1422, July 2012. URL: <http://dx.doi.org/10.1109/TPAMI.2011.239>, doi:10.1109/TPAMI.2011.239.
- [3] G. Nebehay and R. Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision*. IEEE, March 2014.
- [4] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4), December 2006. URL: <http://doi.acm.org/10.1145/1177352.1177355>, doi:10.1145/1177352.1177355.
- [5] K Grauman and B Leibe. *Visual Object Recognition*. Synthesis lectures on artificial intelligence and machine learning. Morgan & Claypool Publishers, 2010. URL: <https://books.google.pt/books?id=1AQGBvdm3UsC>.
- [6] C. Stiller and R. Suntrup. Parametric object motion estimation. In *Singapore ICCS/ISITA '92. 'Communications on the Move'*, pages 633–637 vol.2, Nov 1992. doi:10.1109/ICCS.1992.255187.
- [7] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):564–577, May 2003.
- [8] J. Shi and C. Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.
- [9] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, May 2002. URL: <http://dx.doi.org/10.1109/34.1000236>, doi:10.1109/34.1000236.
- [10] C. Lee. An accessible introduction to mean-shift, 2015. URL: <http://sociograph.blogspot.pt/2011/11/accessible-introduction-to-mean-shift.html>.
- [11] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149 vol.2, 2000. doi:10.1109/CVPR.2000.854761.
- [12] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of Cal. Math. Soc.*, 35(1):99–109, 1943.

- [13] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. URL: <http://dl.acm.org/citation.cfm?id=1623264.1623280>.
- [14] C Tomasi and T Kanade. *Detection and Tracking of Point Features*. Shape and motion from image streams. School of Computer Science, Carnegie Mellon Univ., 1991. URL: <https://books.google.pt/books?id=20wpSQAACAAJ>.
- [15] R. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, 1960.
- [16] J. Liu and R. Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93:1032–1044, 1998.
- [17] I. Reid and H. Term. Estimation ii, 2001 (Last accessed May 5, 2015). URL: <http://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf>.
- [18] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Trans. Sig. Proc.*, 50(2):174–188, February 2002. URL: <http://dx.doi.org/10.1109/78.978374>, doi:10.1109/78.978374.
- [19] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman filter : particle filters for tracking applications*. Artech House, Boston, London, 2004. URL: <http://opac.inria.fr/record=b1102164>.
- [20] M. Rochoux. Bayesian inference for front-tracking problems - 2013 ipdo conference, 2014 (Last accessed June 3, 2015). URL: <http://www.slideshare.net/mrochoux/bayesian-inference-for-fronttracking-problems>.
- [21] M. Isard and A. Blake. Condensation—conditional density propagation for visual tracking. *Int. J. Comput. Vision*, 29(1):5–28, August 1998. URL: <http://dx.doi.org/10.1023/A:1008078328650>, doi:10.1023/A:1008078328650.
- [22] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society. URL: <http://dl.acm.org/citation.cfm?id=850924.851523>.
- [23] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society. URL: <http://dx.doi.org/10.1109/CVPR.2005.177>, doi:10.1109/CVPR.2005.177.
- [24] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vi-*
sion, 60(2):91–110, November 2004. URL: <http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94>, doi:10.1023/B:VISI.0000029664.99615.94.
- [25] D. Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image, 2004. URL: <http://www.google.com/patents/US6711293>.

- [26] Fixational. Opencv vs matlab, 2014 (Last accessed June 4, 2015). URL: <http://blog.fixational.com/post/19177752599/opencv-vs-matlab>.
- [27] OpenCV. Android, 2015 (Last accessed June 5, 2015). URL: <http://opencv.org/platforms/android.html>.
- [28] G. Nebehay, B. Micusik, C. Picus, and R. Pflugfelder. Evaluation of an online learning approach for robust object tracking. Technical Report AIT-DSS-TR-0279, AIT Austrian Institute of Technology, March 2011.
- [29] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *ICPR*, pages 2756–2759. IEEE Computer Society, 2010. URL: <http://dblp.uni-trier.de/db/conf/icpr/icpr2010.html#KalalMM10>.
- [30] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001, Kauai, HI, USA*, pages 511–518, 2001. URL: <http://doi.ieeecomputersociety.org/10.1109/CVPR.2001.990517>, doi:10.1109/CVPR.2001.990517.
- [31] F. Chang, C. Chen, and C. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Comput. Vis. Image Underst.*, 93(2):206–220, February 2004. URL: <http://dx.doi.org/10.1016/j.cviu.2003.09.002>, doi:10.1016/j.cviu.2003.09.002.
- [32] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007. doi:10.1109/CVPR.2007.383123.
- [33] V. Lepetit, P. Laguerre, and P. Fua. Randomized trees for real-time keypoint recognition. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 775–781 vol. 2, June 2005. doi:10.1109/CVPR.2005.288.
- [34] M. Blaschko. Branch and bound strategies for non-maximal suppression in object detection. In Y. Boykov, F. Kahl, V. Lempitsky, and F. Schmidt, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 6819 of *Lecture Notes in Computer Science*, pages 385–398. Springer, 2011. URL: http://dx.doi.org/10.1007/978-3-642-23094-3_28, doi:10.1007/978-3-642-23094-3_28.
- [35] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *Computer Journal*, 26(4):354–359, 1983.
- [36] Z. Kalal, J. Matas, and K. Mikolajczyk. P-n learning: Bootstrapping binary classifiers by structural constraints. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 49–56, June 2010. doi:10.1109/CVPR.2010.5540231.
- [37] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010.
- [38] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV, ECCV'10*, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=1888089.1888148>.

- [39] R. Xu and D. Wunsch. Survey of clustering algorithms. *Trans. Neur. Netw.*, 16(3):645–678, May 2005. URL: <http://dx.doi.org/10.1109/TNN.2005.845141>, doi:10.1109/TNN.2005.845141.
- [40] G. Nebehay. Opentld, 2015 (Last accessed June 7, 2015). URL: <https://github.com/gnebehay/OpenTLD>.
- [41] G. Nebehay. Cppcmt, 2015 (Last accessed June 7, 2015). URL: <https://github.com/gnebehay/CppMT>.
- [42] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010. URL: <http://dx.doi.org/10.1007/s11263-009-0275-4>, doi:10.1007/s11263-009-0275-4.
- [43] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):810–815, June 2004. doi:10.1109/TPAMI.2004.16.
- [44] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, pages 233–240, New York, NY, USA, 2006. ACM. URL: <http://doi.acm.org/10.1145/1143844.1143874>, doi:10.1145/1143844.1143874.