

AMERICAN UNIVERSITY OF BEIRUT

REAL TIME QOS CONTROL FOR VIDEO TRACKING

by  
LEA S. BOUTROS

A thesis  
submitted in partial fulfillment of the requirements  
for the degree of Master of Science  
to the Department of Computer Science  
of the Faculty of Arts and Sciences  
at the American University of Beirut

Beirut, Lebanon  
May 2014


AMERICAN UNIVERSITY OF BEIRUT

REAL TIME QOS CONTROL FOR VIDEO TRACKING

by  
LEA S. BOUTROS

Approved by:

  
\_\_\_\_\_  
Dr. Mohamad Jaber, Assistant Professor  
Computer Science  
Advisor

  
\_\_\_\_\_  
Dr. Wassim El Hajj, Assistant Professor  
Computer Science  
Member of Committee

  
\_\_\_\_\_  
Dr. Shady Elbassuoni, Assistant Professor  
Computer Science  
Member of Committee

Date of thesis defense: May 9, 2014

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Boutros Lea Salim  
Last First Middle

Master's Thesis                       Master's Project                       Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project**, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

Boutros May 19, 2014  
Signature Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

## ACKNOWLEDGEMENTS

I would like, first of all, to thank my advisor Dr. Mohamad Jaber who played a major role in the accomplishment of my thesis.

I would also like to express my gratitude to the committee members, whose help and advices were vital and essential. Special thanks to Dr. Wassim El Hajj for his precious help.

In addition, a thank you to Dr. George Turkiyyah, for the help and effort he has put during my master years at AUB. Also, my special appreciation goes to Mr. Mike Hamam, always present through both good and bad times, he was always ready and happy to offer his help in any way.

Finally I can't find words to describe how thankful I am for the support my family gave me. Without the constant encouragement and the strength they gave me all along, none of this would be happening.

# AN ABSTRACT OF THE THESIS OF

Lea S. Boutros      for      Master of Science  
Major: Computer Science

Title: Real Time QoS Control For Video Tracking

In this thesis, we propose a method to match a set of logos in a video in real time without degrading the quality of the video. The matching algorithm should take into account frame rate standard so that we avoid skipping frames which drastically reduces the quality of a video.

The contribution of the thesis is two-fold. First, we propose a solution based on fine grain Quality of Service (QoS) control so that to increase predictability of execution times. Our method allows adapting matching logo algorithm by adequately setting quality level parameters for it. The objective of the quality management policy is to meet QoS requirements while maximizing the utilization of available time budget. This allows us to match a set of logos by adapting the quality of the matching algorithm depending on the available deadline. Depending on the progress of the computation (actual time), our method uses a quality manager that chooses the next quality level parameter. Second, we use multithreading in order to parallelize matching of different set of logos, and hence maximize the number of logos that could be matched within the deadline.

Our method is fully implemented using C++ and OpenCV library. We present experimental results showing that using our method we can match more logos while respecting the frame rate of a video.

# CONTENTS

ACKNOWLEDGEMENTS . . . . .	iii
ABSTRACT . . . . .	iv
LIST OF FIGURES . . . . .	vii
LIST OF TABLES . . . . .	ix
1 INTRODUCTION . . . . .	1
1.1 Background . . . . .	2
1.2 Problem Statement . . . . .	2
1.3 Solution Proposed . . . . .	3
1.4 Thesis Outline . . . . .	3
2 PRELIMINARIES . . . . .	5
2.1 Matching Algorithm . . . . .	5
2.1.1 Keypoints Extraction . . . . .	5
2.1.2 Descriptors Extraction . . . . .	10
2.1.3 FLANN Based Matcher . . . . .	12
2.1.4 Homography Matrix . . . . .	13
2.2 Quality of Service . . . . .	15
3 RELATED WORK . . . . .	17
3.1 Matching Algorithm . . . . .	17
3.2 QoS controller . . . . .	19
3.3 Our Contribution . . . . .	21
4 MATCHING ALGORITHM . . . . .	22
4.1 Introduction . . . . .	22
4.2 Algorithm . . . . .	22
4.3 Example . . . . .	26
4.4 Conclusion . . . . .	29
5 QUALITY CONTROL . . . . .	30
5.1 Introduction . . . . .	30
5.2 Quality Selection . . . . .	31
5.3 Algorithm . . . . .	35
5.4 Experimental Results . . . . .	40
6 PARALLELIZE OUR ALGORITHM USING PTHREADS . . . . .	44
6.1 Introduction . . . . .	44
6.2 Algorithm . . . . .	45
6.3 Experimental Results . . . . .	47

7 CONCLUSION . . . . .	52
------------------------	----

# LIST OF FIGURES

2.1	An image convolved with a Gaussian function under different scales. .	7
2.2	Scale space of an image. We can see 3 different octaves containing the convolved images with different parameter. The size of the image decreases for every octave. At the right we can see the difference of Gaussian of these images. . . . .	9
2.3	Images represented by their pixel values . . . . .	11
2.4	Entry value of pixel $(x, y)$ in the original image $I$ and the integral image $I_\Sigma$ . . . . .	11
2.5	Controller architecture . . . . .	15
3.1	The DoG applied for every two consecutive images convolved with a Gaussian function with scale $k\sigma$ . . . . .	18
3.2	Three consecutive DoG images. We take the candidate interest point (marked with $\times$ ) and take its 26 neighbors (marked with circles) in the same scale and in adjacent scales $(3 \times 3)$ and check if it is a maxima or minima. . . . .	18
3.3	Instead of iteratively reducing the image size (left), the use of integral images allows the up-scaling of the filter at constant cost (right). . . .	19
4.1	We want to match these two images. The goal is to find the logo in the commercial image. . . . .	26
4.2	Red circles around keypoints found in the commercial image and the logo . . . . .	27
4.3	Matches between the frame and the logo. . . . .	27
4.4	Matches (denoted in red) between the keypoints in the frame and the ones in the logo. . . . .	28
4.5	The logo is found in the commercial image. . . . .	29
5.1	The better the quality is, the more time it needs to execute. . . . .	30
5.2	Coca-cola commercial with different determinant of hessian threshold. We notice that when we increase the threshold, we get less keypoints and therefore we miss the logo in the image. . . . .	32
5.3	Controller architecture . . . . .	37
5.4	Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 3 logos. . . . .	41
5.5	Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 6 logos. . . . .	42
5.6	Screenshot of the resulting video using the quality manager . . . . .	43
5.7	Screenshot of the resulting video using the fixed quality $q_2$ . . . . .	43
6.1	Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 3 logos using one thread. . . . .	48



6.2 Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 6 logos using 2 threads. . . . . 49

6.3 Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 6 logos using 1 thread. . . . . 49

6.4 Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 9 logos. . . . . 50

6.5 Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 12 logos. . . . . 51

## LIST OF TABLES

5.1	Table representing the average execution time (in ms) of the matching algorithm with different threshold values and decreasing the number of keypoints. . . . .	33
5.2	Table representing the average execution time (in ms) of the matching algorithm with different threshold values and decreasing the number of keypoints. . . . .	34
5.3	Table representing the average execution time (in ms) of the matching algorithm with different threshold values and decreasing the number of keypoints. . . . .	34
5.4	Table describing the different quality values chosen for our algorithm.	35
6.1	Table representing the average execution time of a frame matched with the specified number of logos using 1, 2, 3 or 4 threads. . . . .	48

# CHAPTER 1

## INTRODUCTION

The technology revolutionized the world we live in since the creation of the first computer in 1946. Back then, the computer was over 30 tons and had only one functionality. Half a century later, this computer can fit in your pocket and it provides a multitude of functionalities. Some might say that technology took over our humanity by introducing this new virtual world. This might be true, however it should be a mean to make us move forward. Information should come to us when needed, and with the introduction of augmented reality (AR) we are closer to this goal. AR is a technology which consists in overlaying digital information onto the real world. Its main goal is to make our surrounding environment interactive by adding useful information to it. The applications are quite wide and appealing. Tourism is one example where such technology becomes handy and desirable, because when you are in a new environment, you are more curious and eager for information. For instance, if we are standing in front of a monument and we want to know its history we could use the camera of our phone. The needed information will be added to the real time video that we are seeing. What will actually happen is that the camera will take snapshots of the monument and send them to a server. A matching algorithm will recognize the monument and give back the respective information that will be displayed on the image itself. Masking logos in a video is another major example where augmented reality can be used. This system needs to recognize and localize logo in real time to be able to mask them while not affecting the quality of the video. The augmented information in this case is the application of a blurred mask on the logo found.

Augmented reality is the future of the virtual world because its main purpose is to enhance our perception of the real world. A lot of work has been done in this

field such as mobile and desktop applications.

## 1.1 Background

Google goggles [Google] is a snapshot recognition system, based on image search. By taking a photo of a well-known landmark, you obtain information about it. However, this system does not overlay the information on the picture.

Layar Quintin and Maarten [2009] is a phone application that uses the GPS location to retrieve information about the area we are located in. For instance, we can retrieve the restaurants around us. We can also look at a building and know its history from Wikipedia. Wikitude [Herdina et al.] is a GPS based MAR system like layar. These two systems give information about the users surroundings by augmenting them on the camera view. These techniques rely on the phones sensor data.

A 3D mobile augmented reality system for urban scenes is presented in Wu et al. [2011]. In order to get accurate results, this system does not only use the mobiles sensor data, it also leverages the visual information by extracting the features from the live video frames and compare them to the images database. Additionally, it augments the retrieved data with the correct perspective by matching the query to the 3D models in the database.

## 1.2 Problem Statement

We aim to implement a system that recognizes and localizes logos in a video and provides relevant information to the user, such as advertisement or blurred masks. This application aims at obtaining and augmenting the information on the video.

On the first hand, a video runs at a rate of 30 frames per second, i.e. every frame is displayed for 33 ms. Therefore, the matching process should take less than 33 ms otherwise the quality of the video will drastically be reduced. In fact, the video will still be composed of the same number of frames but, the display time will

be bigger. Moreover, our goal is to match a video to more than one logo within the same deadline, this makes our task more challenging.

On the other hand, execution times for matching algorithm may considerably vary over time as they depend on the size and quality of the logo and frame. Furthermore, non predictability on the underlying platform is an additional factor of uncertainty.

### **1.3 Solution Proposed**

In order to match a maximum number of logos without degrading the quality of a video, we propose a solution based on fine grain Quality of Service (QoS) control so that to increase predictability of execution times. Our method allows adapting matching logo algorithm by adequately setting quality level parameters for it. The objective of the quality management policy is to meet QoS requirements while maximizing of the utilization of available time budget. This allows to match a set of logos by adapting the quality of the matching algorithm depending on the available deadline. Depending on the progress of the computation (actual time), our method uses a quality manager chooses the next quality level parameter.

Moreover, our method uses multithreading in order to parallelize matching of different set of logos. In this thesis we will discuss in detail how we merge those techniques in order to match the maximum number of logos without reducing the quality of a video (i.e., skipping some frames).

Our method is fully implemented using C++ and OpenCV library [2006] which is an open source library for computer vision. We present experimental results using different case studies.

### **1.4 Thesis Outline**

The rest of the thesis consists of 6 chapters. In Chapter 2 we present some preliminaries. Then, in Chapter 3 we discuss related works. In Chapter 4, we

present a matching algorithm based on SURF features. In Chapter 5, we present the first solution based on fine grain Quality of Service control to manage the execution times. In Chapter 6, we present the second solution based on multithreading in order to parallelize matching of different set of logos. Finally, Chapter 7 draws conclusion and future work.

# CHAPTER 2

## PRELIMINARIES

### 2.1 Matching Algorithm

Finding correspondences between two images has a large field of applications. Image matching is one of them. There are three major parts to find point correspondences between two images. First, we have to find “interest points”. Then we take the points that lie in the neighborhood of every interest point and compute a vector, it is called the “descriptor”. Finally, we match the descriptors of different images based on Euclidean distance.

The matching Algorithm that we used is based on SURF (Speed-Up Robust Features) [Bay et al., 2008] features.

#### 2.1.1 *Keypoints Extraction*

Keypoints Extraction is one of the major parts of image recognition. Moreover, it has a wide range of other applications (e.g., face recognition, motion tracking and 3D reconstruction). Keypoints extraction requires to identify the major characteristics of an interest point. An interest point is a point in an image that is recognizable in any variation of this image or in a slight change of perspective. A feature detector is responsible for extracting these points. For this, the feature detector should be repeatable, i.e. it should find the same interest points under different views.

In an image we can find four types of regions and just two of them are unique enough to be distinguished out of their environments. For example, a region centered around a point in a flat area cannot be considered as a feature, because it is similar to its environment. So, if we move the rectangular area around this point and compare it to the one that we had before, we can clearly see that they are similar. If we take an edge, for example, we can notice the contrast between the two sides of the

edge. However, if we move the region along the edge, we will notice that there is no difference. Corners and blobs are considered to be good features because they are unique. If we move the rectangular area around them and compare them we find a really big difference.

Consider a small rectangular area of pixels around a candidate feature  $(x_0, y_0)$ . Suppose that we move the box by a vector  $(u, v)$  so the point  $(x_0, y_0)$  becomes  $(x_0 + u, y_0 + v)$ . For the feature to be a good one, we should find a difference between the 2 boxes. So we compute the sum of squared differences of every pixel (see equation 2.2).

$$C(u, v) = \sum_{(x,y) \in \text{Box}} (I(x, y) - I(x + u, y + v))^2 \quad (2.1)$$

This cost function should be large for all  $(u, v)$ , for  $(x_0, y_0)$  to be a good feature. If we expand  $C(u, v)$  in a Taylor series at  $(u, v) = (0, 0)$ , we get the following equation.

$$\begin{aligned} C(u, v) &\approx \sum_{(x,y) \in \text{Box}} (I(x, y) - (I(x, y) + u \frac{\partial I}{\partial x} + v \frac{\partial I}{\partial y}))^2 \\ &= \sum u^2 \frac{\partial I^2}{\partial x} + 2uv \frac{\partial I}{\partial x} \frac{\partial I}{\partial y} + v^2 \frac{\partial I^2}{\partial y} \\ &= \begin{bmatrix} u \\ v \end{bmatrix}^T H \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned} \quad (2.2)$$

with H being the Harris Matrix (see equation 2.4).

$$H = \begin{bmatrix} \sum \frac{\partial I^2}{\partial x} & \sum \frac{\partial I}{\partial x} \frac{\partial I}{\partial x} \\ \sum \frac{\partial I}{\partial x} \frac{\partial I}{\partial x} & \sum \frac{\partial I^2}{\partial y} \end{bmatrix} \quad (2.3)$$

We want the cost function to be high for the candidate feature to be a good one. This means that we want the eigenvalues of  $H$  to be large. However, for computation reasons, we compute  $\det(H) - K \text{trace}(H)$ . It is small when one or both eigenvalues



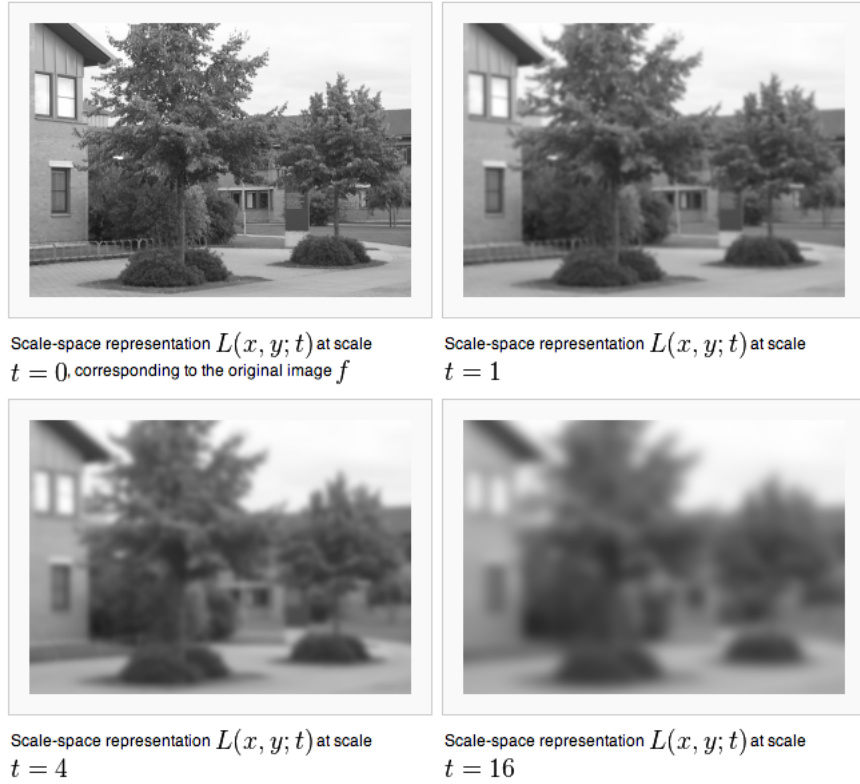


Figure 2.1: An image convolved with a Gaussian function under different scales.

are small, and it is high otherwise. However, we get better results when smoothing the image, especially when it is centered around the candidate interest point since we give it more weight.

$$H = \begin{bmatrix} \sum \frac{\partial I^2}{\partial x} & \sum \frac{\partial I}{\partial x} \frac{\partial I}{\partial x} \\ \sum \frac{\partial I}{\partial x} \frac{\partial I}{\partial x} & \sum \frac{\partial I^2}{\partial y} \end{bmatrix} * weight \quad (2.4)$$

To smooth the image we convolute it with a Gaussian function centered at the candidate point  $(x_0, y_0)$  with a scale  $\sigma$ .

$$L(x, y, \sigma) = g(x, y, \sigma) * I(x, y) \quad (2.5)$$

Thus the weight function used in this case is the Gaussian function and we get the following equation:

$$H = \begin{bmatrix} \sum \frac{\partial I^2}{\partial x} & \sum \frac{\partial I}{\partial x} \frac{\partial I}{\partial x} \\ \sum \frac{\partial I}{\partial x} \frac{\partial I}{\partial x} & \sum \frac{\partial I^2}{\partial y} \end{bmatrix} * g(x, y, \sigma) \quad (2.6)$$

### Derivative of Gaussian

We use the derivative of Gaussian to smooth the image and give more weight to the candidate feature point.

$$\frac{\partial I}{\partial x} \approx I(x, y) * \frac{\partial G(x, y, \sigma)}{\partial x} \quad (2.7)$$

We use Gaussian derivatives as feature detector because of its rotation and translation invariance.

$$L_x^m L_y^n(x, y, \sigma) = \frac{\partial^{m+n}}{\partial x^m \partial y^n} g(x, y, \sigma) * f(x, y) \quad (2.8)$$

### Hessian of Gaussian

$$\mathcal{H}(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix} \quad (2.9)$$

In order to get more precise results we assume that a point is a keypoint if it is still recognizable under different modifications. For this, we construct the scale space of the image.

### Scale Space

A scale space of an image, in computer vision, is a collection of variation of this image. It is done by repeatedly smoothing the image, i.e. by applying a linear (Gaussian) filter with different parameters  $\sigma$  (see equation 2.5). The image  $I(x, y)$  is subjected to a set of linear transformation  $L(x, y, \sigma)$  with varying scale  $\sigma$ . So for

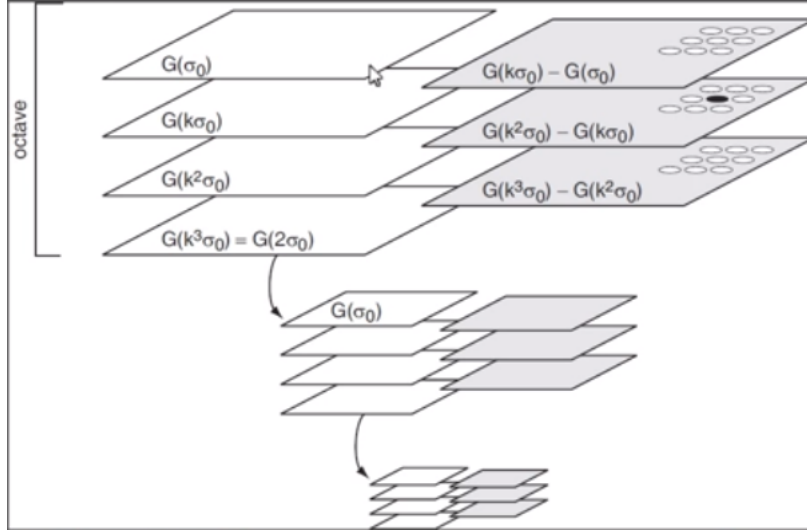


Figure 2.2: Scale space of an image. We can see 3 different octaves containing the convolved images with different parameter. The size of the image decreases for every octave. At the right we can see the difference of Gaussian of these images.

a given fixed scale  $\sigma$ , the new image will be a convolution of the image  $I(x, y)$  with the Gaussian function  $g(x, y, \sigma)$  (see equation 2.10)

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma} e^{-(x^2+y^2)/2\sigma} \quad (2.10)$$

A lot of methods have been used to create the scale space of an image, however, the Gaussian kernel is the most appropriate filter for image recognition. In fact, it is scale invariant, shift invariant and more importantly, it doesn't create inexistent structures when passing from a fine image to any coarser one.

The scale space is composed of several octaves, each of which contains the convolved image with parameter  $\{\sigma_0, k\sigma_0, k^2\sigma_0, k^3\sigma_0\}$ . To keep the same Gaussian parameters for every octave, it has been shown that reducing the resolution of the image is equivalent to changing the parameters. Thus, for every octave the original image size is decreased. This is done for computational purposes, because convolving a smaller image is done faster. Lowe used this method to extract its feature points. However, in 2008, with the introduction of SURF [Bay et al., 2008], it has been shown that the scale space can be computed with the original size in a constant

execution time using integral images.

### Integral Images

Integral image, which is also called summed area table, was introduced in the computer vision field in 2001 by Viola and Jones [Viola and Jones, 2001]. Integral images are used to compute the sum of pixel values in a given image or in a rectangular area of this image, in a quick way. It computes it in four memory accesses independently of the size of the image.

Suppose that we have an image  $I$  represented as a matrix where every entry  $(x, y)$  contains the pixel value at this corresponding position, as shown in Figure 2.3a. The integral image  $I_\Sigma$  is a matrix where every entry  $(x, y)$  contains the sum of the pixel values of the left top corner of the original image  $I$  including the value at the  $(x, y)$  position (see figure 2.4a). Figure 2.3b is the integral image of figure 2.3a.

$$I_\Sigma(x, y) = \sum_{i=0}^x \sum_{j=0}^y I(i, j) \quad (2.11)$$

We can compute the integral image in only one pass over the original image. So the previous equation can be replaced by equation 2.12.

$$I_\Sigma(x, y) = I(x, y) + I_\Sigma(x - 1, y) + I_\Sigma(x, y - 1) - I_\Sigma(x - 1, y - 1) \quad (2.12)$$

For any rectangular area in the integral image, it takes only 3 operations to find the sum of its intensity. So, the integral image reduces the computation time to  $\mathcal{O}(1)$ , which is independent of its size.

#### **2.1.2 Descriptors Extraction**

The descriptors are used to describe the interest points. In SURF, the descriptors describe the distribution of the intensity around the interest point. They are

4	3	6	3
5	6	2	2
9	5	2	3
4	1	6	5

(a) Original Image  $I$

4	7	13	16
9	18	26	31
18	32	42	50
22	37	53	66

(b) Integral Image  $I_\Sigma$

Figure 2.3: Images represented by their pixel values

		$I(x, y)$	

(a) Original image  $I$

	$I_\Sigma(x-1, y-1)$	$I_\Sigma(x, y-1)$	
	$I_\Sigma(x-1, y)$	$I_\Sigma(x, y)$	

(b) Integral image  $I_\Sigma$

Figure 2.4: Entry value of pixel  $(x, y)$  in the original image  $I$  and the integral image  $I_\Sigma$

extracted using the first order Haar wavelets.

### Orientation Vector

This step is done to ensure that our descriptor is rotation invariant. For each interest point we want to find its orientation vector. For this, we take a set of sliding windows around the point and calculate the Haar wavelet responses in both x and y directions. We add the horizontal and vertical responses. Then, the sum of these two responses gives us the local orientation vector of this sliding window. After repeating this process for all windows around this point, we compare the resulting vectors. The longest one defines the orientation vector of this interest point.

### Extraction of Descriptor

First, we take a squared region around the interest point and oriented along the vector that we previously computed. Then, we split this region into 4x4 boxes. For every sub-region, we calculate the Haar wavelet responses in both directions horizontally and vertically that we call  $dx$  and  $dy$  respectively. We also compute the absolute values of these responses  $|dx|$  and  $|dy|$ . Hence, each sub-region has a 4 dimensional vector  $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$ .

The squared regions around the interest point is composed of  $4 \times 4$  such sub-regions, i.e. 16 boxes that can be described by  $v$ . So, we concatenate  $v$  for all the 16 sub-regions and we get a 64 dimensional vector. This vector is the descriptor of the interest point.

#### **2.1.3 FLANN Based Matcher**

FLANN [Muja and Lowe, 2012] is a library for Fast Approximate Nearest Neighbor algorithms. This library encloses several matching algorithms with different data structures. We are interested in Kd-trees for image matching techniques. A Kd-tree is a balanced binary tree where every node has k dimensions. The data is

split according to the axes (x, y, z, ...). Assume that we have 4 dimensional data, every node is of the form  $(x_1, x_2, x_3, x_4)$ . We first split the data according to  $x_1$ , i.e. the nodes are compared according to the values of  $x_1$ . The elements that are less than the node are in the left subtree and the elements that are higher than the node are in the right subtree. Then this process is repeated until all the data has its position in the tree. At every level of the tree the data is classified according to one of the 4 dimensions.

Kd-trees are efficient in low dimensions. In order to operate in high dimensions we have to use multiple randomized Kd-tree instead.

We construct the trees for one of the images, then we compare every descriptor to the trees and find the best two matches. For all descriptors in the query image, we have the two best matching descriptors of the second image. To know if a match is a good one, it should be unique. This means that the distance between the best match and the query descriptor should be far from the distance between the second best and the query.

#### 2.1.4 Homography Matrix

A Homography matrix can be considered as a projection matrix that projects a point from the source image to its matching point in the destination image. It is composed of a translation vector  $t = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$  and a rotation matrix  $R = \begin{pmatrix} R_1 & R_2 \\ R_3 & R_4 \end{pmatrix}$ . Suppose that we have a point  $(x, y)$  in the source image, then its corresponding point  $(X, Y)$  in the destination image can be found solving the following set of equation.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.13)$$

with  $\begin{cases} x = a/c \\ y = b/c \end{cases}$

In our case, we do not have the homography matrix, however, we have the two images that we are trying to match and the set of matching points of these images. Using the set of matching points we need to find the homography matrix which transforms the first set of points into their corresponding points in the second one.

The Homography matrix is a 3x3 matrix, so it has 9 entries. The system has 9 unknowns so we need 3 pair of points (9 equations) in order to solve it and find the matrix. Some of the pairs of point are false positive matches, hence we cannot randomly choose three points out of the set to solve the system. Therefore, we use the RANSAC (RANDOM SAmple Consensus) [Fischler and Bolles, 1981] algorithm. This consists in choosing 3 pairs of points randomly and in computing the Homography matrix.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.14)$$

Known Unknown Known

Then, using this matrix and the set of points coming from the original image, we recalculate all the matching points.

$$\begin{bmatrix} a' \\ b' \\ c' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.15)$$

Unknown Known Known

After getting the new set of points, we calculate the difference between the old points  $(x, y)$  and the new ones  $(x', y')$ . The sum of squared differences is the error that this homography produces. We store both the error and its corresponding



matrix. We repeat this whole process a multitude of times and then the Homography matrix with the smallest error is selected as the best one. All pairs of points that do not give a good result when multiplied by H are considered as outliers. Otherwise, we call them inliers.

## 2.2 Quality of Service

In this thesis we present a method that applies SURF matching algorithm under a controlled software. This is done to control the execution time of our parameterized system. The controlled software is composed of the initial algorithm and a controller. The main purpose of the controller is to control the execution of a cycle and then choose the appropriate following action to run with its quality. To do so, it has two main components: the quality manager and the scheduler. In our case we do not need the scheduler, since our algorithm is composed of a single sequence of cycles, without different options.

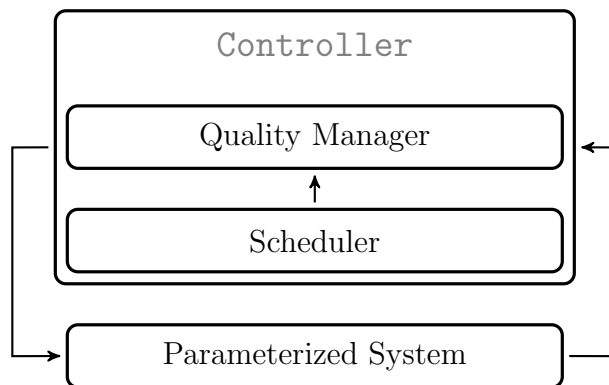


Figure 2.5: Controller architecture

Let  $Q = \{q_1, q_2, q_3, \dots, q_m\}$  be the set of all qualities such that  $q_1$  is the best quality. Suppose that we want to run our algorithm  $k$  times without exceeding the deadline  $D_1$ . The main purpose is to find the best combination of qualities that allows us to run  $k$  times our program within the deadline. Before executing the  $n^{th}$  cycle, we need to know which quality is going to be used. The controller is the responsible for finding the suitable quality according to the remaining time. First,

the quality manager calculates the new deadline  $D_n$  which is the remaining time to run the  $k - n + 1$  cycles. It takes as parameter the old deadline and the execution time of the previous cycle.

$$D_n = D_{n-1} - E_{n-1} \quad (2.16)$$

After getting the new deadline, it should find the best quality that can be used in the following cycle such that all remaining cycles can still be executed with the lowest quality without exceeding the deadline. For this, the quality manager assumes that the following  $k - n$  cycles will be executed with the lowest quality  $q_m$  and computes the average time needed to do so. Then, it subtracts this time from the deadline  $D_n$  of the current cycle. This difference represents the time available to execute the  $n^{th}$  cycle and it is used to choose the best quality which satisfies it.

# CHAPTER 3

## RELATED WORK

### 3.1 Matching Algorithm

Matching two images has a wide range of applications. Therefore, lot of work has been done in this area so that all aspects can be covered. There are several methods to match images depending on their nature (face, hand, logo, ...). In our case we are working on logo detection and recognition in a video. However, our detector should be scale and rotation invariant. Most importantly the features should be invariant to illumination or noise.

#### *SIFT*

The Scale Invariant Feature Transform (SIFT) has been introduced by David G. Lowe in 1999 [Lowe 1999]. The Harris corner detector is very sensitive to changes in image scale, so it not efficient for matching images of different sizes. The first work done by Lowe [1999] extended the previous feature detector to insure scale invariance. This method works primarily with the difference of Gaussian (DoG) which is an approximation of the Laplacian. Previous methods used to work with the Laplacian of Gaussian (which is described in the preliminaries chapter).

In figure 3.1, we can see the constructed scale space of the convoluted images by the Gaussian function with different scales  $k\sigma$ , for every octave. Then for every two consecutive images the DoG has been computed. The next step is to find the local extrema. For every point we need to check if it is a maxima or minima among its 26 neighbors (see figure 3.3). Only the points that are found to be local extrema are selected.

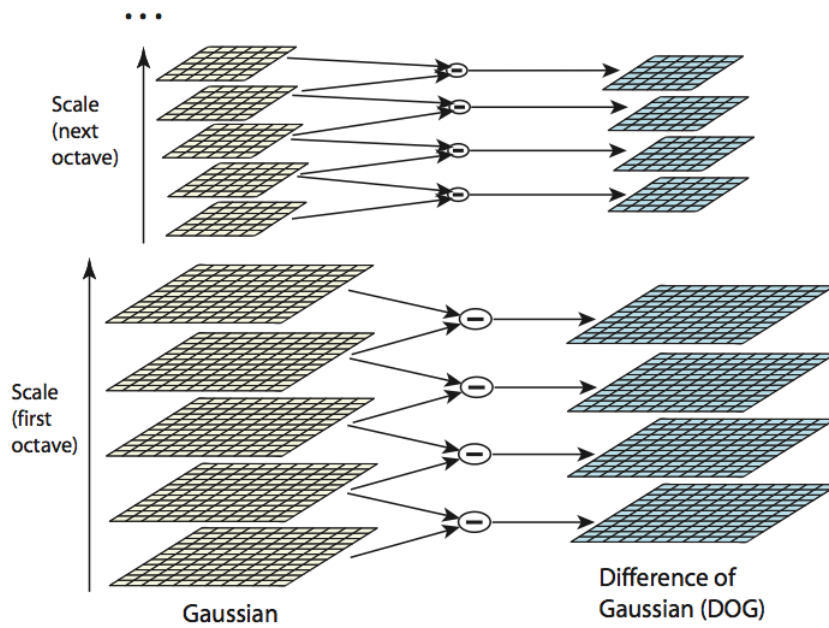


Figure 3.1: The DoG applied for every two consecutive images convolved with a Gaussian function with scale  $k\sigma$

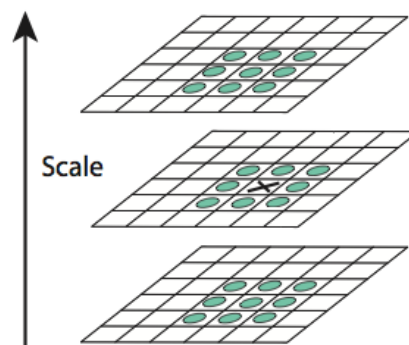


Figure 3.2: Three consecutive DoG images. We take the candidate interest point (marked with  $\times$ ) and take its 26 neighbors (marked with circles) in the same scale and in adjacent scales ( $3 \times 3$ ) and check if it is a maxima or minima.

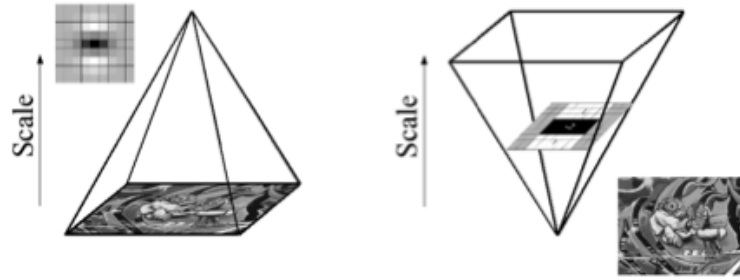


Figure 3.3: Instead of iteratively reducing the image size (left), the use of integral images allows the up-scaling of the filter at constant cost (right).

### ***SURF***

Speed-Up Robust Features (SURF) has been introduced in 2008 Bay et al. [2008]. Its goal is to find invariant features in a faster way without losing performance. The feature detector is based on the hessian matrix.

$$H(x, y, \sigma) = \begin{bmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{yx}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{bmatrix} \quad (3.1)$$

SURF creates its scale space without resizing and reducing the resolution of the images as going up the pyramid. On the contrary it keeps all images of the same original size, since it uses integral image which allows a faster computation (around constant time) of the sum of the pixel in a rectangular box. So it computes a box-filter approximation of the second order Gaussian partial derivatives Bay et al. [2008].

The two algorithms have been tested and it has been shown in Panchal et al. [2013] that SURF is faster than SIFT. In fact it detects less feature points and leads to a faster matching process.

### **3.2 QoS controller**

Wust et al. [2004] propose two solutions to achieve a soft real-time execution for high quality video processing that are based on a Markov decision process and

reinforcement learning.

Buttazzo et al. [1998] describe an elastic task model where every task is represented as a spring with elastic coefficients. A task can change its coefficients in order to control its execution time and the other tasks will adapt so that the time period is respected.

Koren and Shasha [1996] deal with tasks where the deadline can be missed considering that most of the deadlines are met. They proved that using skip techniques optimally is NP-hard. They worked with two skip-over algorithms: Earliest Deadline First (EDF) and Rate Monotonic scheduling (RM), setting some periodic bounds. EDF is used for dynamic scheduling whereas RM is used for static scheduling, where we have complete knowledge about the properties and execution time of the tasks.

Rajkumar et al. [1997] describe a QoS based Resource Allocation Model (Q-RAM). The model is a framework containing several applications that can be run with different level of qualities, depending on the available resources. The main purpose is to assign resources to the different applications such that the overall utility is maximized, while minimizing the need of every application.

Combaz et al. [2005b] propose a method where the quality levels can be set adequately so that the following QoS requirements are respected:

- Safety - No deadline is missed
- Optimality - Maximize the available time
- Smoothness of quality levels

In this paper, they tested their results on a MPEG video encoder.

Combaz et al. [2008] improve the results of Combaz et al. [2005b] in two directions: Symbolic quality management and computation of optimal schedule. Symbolic quality management improvements:

- They introduced the use of speed diagrams. The controlled system is represented as a two dimensional graph. One dimension represents the actual time and the other one describes the virtual time.
- For every state in the diagram there are two kind of speed:
  1. Ideal speed: speed if all the remaining actions are run with fixed quality level  $q$ .
  2. Optimal speed: best use of the available time without missing any deadline.

Computation of optimal schedule improvements:

- Create functions that manages uncertainty (due to the difference between average and worst-case execution time) and fall-back ability (due to the difference between average and worst-case execution time for the worst quality).
- They show that for dynamic scheduling, EDF is not an optimal algorithm. Therefore, they proposed two functions to reach optimality using EDF schedules.

### 3.3 Our Contribution

We aim to implement a system that takes a video as input and finds the logos in every frame, in real-time and display the frames. Therefore, we are going to use a combination of (1) SURF features to describe and match our images, (2) the real-time system that we previously described and (3) parallel programming in order to improve real-time execution.

# CHAPTER 4

## MATCHING ALGORITHM

### 4.1 Introduction

The goal of this thesis is to detect logos in a video, which makes the matching algorithm one of its major parts. Our main goal is to match every frame of the video to a database of logos. So, it is not a simple one-to-one matching. There are several ways to solve this issue. The straightforward solution is to match the image to the set of logos one-to-one successively. For simplicity, assume that our database contains one logo. We will describe the algorithm of matching a video to one logo in the following section using SURF features.

### 4.2 Algorithm

The program takes as input a video from the user and should return the video with the detected logos as shown in algorithm 2. We know that a video is a succession of frames. This means that we have to decompose the video into frames and then apply the matching algorithm 1 to every frame.

First, before loading the video, the features of the logo are extracted and the tree is created. This is the first step to be executed because it is computationally inefficient to do it for every frame. This part can be done once for the database since the descriptors and keypoints of the images are fixed. So this part should be run only once and can also be done offline.

Suppose that we extracted offline the  $k$  keypoints from the logo and consequently their  $k$  corresponding descriptors. The keypoints and descriptors are stored in vectors  $K_{Logo}$  and  $D_{Logo}$  respectively.



$$K_{Logo} = \begin{bmatrix} KL_1 \\ KL_2 \\ \vdots \\ KL_k \end{bmatrix} \quad (4.1)$$

$$D_{Logo} = \begin{bmatrix} DL_1 \\ DL_2 \\ \vdots \\ DL_k \end{bmatrix} \quad (4.2)$$

After loading the video, every frame undergoes the same following process as shown in algorithm 1. We first extract the SURF keypoints of the image, i.e. the ‘interest points’ of the image. Then, we extract the descriptor for every keypoint. After getting the descriptors of the frame, they are matched to the logo’s descriptors, and these matches are used to compute the homography matrix.

For a specific frame, we extract the keypoints and descriptors of the frame and store them in  $K_{frame}$  and  $D_{frame}$  respectively.

$$K_{frame} = \begin{bmatrix} K_1 \\ K_2 \\ \vdots \\ K_n \end{bmatrix} \quad (4.3)$$

$$D_{frame} = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_n \end{bmatrix} \quad (4.4)$$

For every descriptors in  $D_{frame}$  we want to find the best 2 matching descriptors from the logo  $D_{Logo}$ . We perform a K-nearest neighbor search for every descriptor

using the index tree that we created. The value of  $K$  is 2 in this case since we want to find the best 2 matches. This is done by calling the OpenCV function “flannIndex.knnSearch()” which will return two matrices. One containing the indices of the two best matches found for every descriptor.

$$indices = \begin{bmatrix} I_{11} \text{ (best match for } D_1) & I_{21} \text{ (2}^{nd} \text{ best match for } D_1) \\ I_{12} \text{ (best match for } D_2) & I_{22} \text{ (2}^{nd} \text{ best match for } D_2) \\ \vdots & \vdots \\ I_{1n} \text{ (best match for } D_n) & I_{2n} \text{ (2}^{nd} \text{ best match for } D_n) \end{bmatrix} \quad (4.5)$$

The second one contains the Euclidean distances,  $d$  and  $d'$ , between every descriptor and its best match and second best match. This means that  $d_1$  is the distance between  $D_1$  and its best matching descriptor in the logo (which is the  $I_{11}$  element in  $D_{Logo}$ ) and  $d'_1$  represents the distance between  $D_1$  and its second best match in the logo (which is the  $I_{21}$  element in  $D_{Logo}$ ).

$$dist = \begin{bmatrix} d_1 & d'_1 \\ d_2 & d'_2 \\ \vdots & \vdots \\ d_n & d'_n \end{bmatrix} \quad (4.6)$$

After getting the distances for all the descriptors, we have to decide which ones are good matches. To do so, for every descriptor we compare the two distances. If the distances are close we consider that the best match is not a good one because it doesn't stand out. If the ratio  $d/d'$  is greater than 0.6 this means that the distances are close, so it isn't a good match. Otherwise, it is a good one, and we store the

descriptor with its best match in a matrix.

$$results = \begin{bmatrix} D_1 & DL_{I_{11}} \\ D_4 & DL_{I_{14}} \\ \vdots & \vdots \\ D_z & DL_{I_{1z}} \end{bmatrix} \quad (4.7)$$

We apply this for all the descriptors and filter them to keep only the best matches. After getting the best matches, we need to construct the homography matrix, which transforms the set of points from the logo to their matches in the frame. To compute the homography matrix we use the “cv::findHomography” function which is based on the RANSAC algorithm and use it to find the corners of the logo found in the frame.

**Input** : Frame image, logo\_keypoints, logo\_descriptors and the trees that we constructed based on the logo descriptors: flannIndex

**Output**: The frame image with the logo detected and found.

- 1 frame\_keypoints  $\leftarrow$  extractKeypoints(frame);
- 2 frame\_descriptors  $\leftarrow$  extractDescriptors(frame);
- 3 [frame\_match logo\_match]  $\leftarrow$   
match(frame\_keypoints, frame\_descriptors, flannIndex);
- 4 H  $\leftarrow$  getHomography(frame\_match, logo\_match);
- 5 dst\_corners  $\leftarrow$  getCorners(src\_corners, H);

**Algorithm 1:** Matching Algorithm for two images (matchImage)

```

Input : Video and a logo
Output: The video with the logo detected and found.
1 logo_keypoints ← extractKeypoints(logo);
2 logo_descriptors ← extractDescriptors(logo);
3 flannIndex ← flannIndex.build(logo_descriptors,
   cv::flann::KDTreeIndexParams(), cvflann::FLANN_DIST_EUCLIDEAN);
4 while not finished do
5     | frame ← video.getframe();
6     | matchImage(frame, logo_keypoints, logo_descriptors, flannIndex);
7     | display(frame);
8 end

```

**Algorithm 2:** Matching Algorithm for a video

### 4.3 Example

For example, we chose a Coca-Cola commercial, where the logo is visible in most of the frames, to test our matching algorithm. We select one particular frame of this video containing the logo and the logo itself, see figure 4.1 and then match these two images using the algorithm that we described in the previous section.



(a) Coca-cola commercial



(b) Coca-cola logo

Figure 4.1: We want to match these two images. The goal is to find the logo in the commercial image.

We extract the keypoints and descriptors for the frame and the logo, as shown in figure 4.2.



(a) Coca-cola commercial with keypoints detected.



(b) Coca-cola logo with keypoints detected.

Figure 4.2: Red circles around keypoints found in the commercial image and the logo

Then we find the best matches between the two images, see figure 4.3 and use them to compute the homography matrix  $H$ .



Figure 4.3: Matches between the frame and the logo.

$$H = \begin{bmatrix} 0 & 0.66 & 246 \\ -0.58 & 0 & 391 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.8)$$

As we can see, the  $3^{rd}$  row in  $H$  is  $(0 \ 0 \ 1)$  which shows that we are working in 2D.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 & 0.66 & 246 \\ -0.58 & 0 & 391 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.9)$$

The system that we get from the previous equation is the following:

$$\begin{aligned} X &= 0 \times x + 0.66 \times y + 246 \\ Y &= -0.58 \times x + 0 \times y + 391 \\ Z &= 0 \times x + 0 \times y + 1 \end{aligned} \quad (4.10)$$

Using this system we can know which matches are the ones satisfying it and we denote them as inliers. We draw the matches between the two images as shown in figure 4.4.



Figure 4.4: Matches (denoted in red) between the keypoints in the frame and the ones in the logo.

We use the homography matrix and the corners of the logo to find their matches in the commercial image.



Figure 4.5: The logo is found in the commercial image.

Matching the commercial frame to the coca-cola logo took 75 ms to execute using a computer which integrates Intel Core i5-2410M @ 2.29 GHz (4 cores).

#### 4.4 Conclusion

The main issue is that we need this algorithm to operate in real-time. We know that a video runs at a rate of 30 frames per second. Every frame is displayed for 33 ms, this is the time available to run the matching algorithm for the following frame. It is nearly impossible to execute this algorithm in 33 ms. However, every 6 consecutive frames in a video can be considered as similar, so we can execute the algorithm once every 6 frames. Thus, saving some time. Therefore, we get  $6 \times 33 = 198$  ms for every execution.

For this particular case, 75 ms per frame is enough. But, how can we insure that we do not cross this deadline when running this algorithm?

# CHAPTER 5

## QUALITY CONTROL

### 5.1 Introduction

The controller is used to control the execution time of our algorithm and to make sure that the deadline  $D = 198$  ms is not being crossed. It is composed of the quality manager which is used as a quality measure. This means that it finds the quality that can be used having a certain amount of time. The main issue in our system is the uncertainty of our algorithm's execution time. Therefore, it is judicious to use QoS, because it makes sure that the deadline is not being crossed.

To do so, we need to understand the notion of quality. We have to find a parameter that varies the quality of the results and the execution time, in the same direction. This means that to get a better result this implies that it takes more time to compute the algorithm. The better the quality is, the more time it needs to execute, as we can see in figure 5.1. The error function used to measure the quality of the result is based on the difference between the features of the original logo and the features of the logo found in the frame.



Good Quality  
Execution Time: 115 ms  
Error Function: 15%

Bad Quality  
Execution Time: 35 ms  
Error Function: 20%

Figure 5.1: The better the quality is, the more time it needs to execute.



## 5.2 Quality Selection

How do we select appropriate qualities for our matching algorithm? First, we need to make sure that our algorithm depends on some parameters that can affect the quality of the results and the execution time. The matching algorithm execution time depends on the size of the image, the number of keypoints extracted, the length of the descriptor vector (64 or 128) and the number of matches.

We decide to decrease the resolution of the image and fix its size. We also adjudicate to set the size of the descriptor vector to be 64. These choices have been made because they favor a speedup in the execution time without reducing the quality of the results. This leaves us with two more factors, the number of keypoints and the number of matches. We cannot control the latter manually, however, it closely depends on the former. So the only characteristic left that affects the running time is the number of keypoints extracted. We can control indirectly the number of interest points detected from an image using the `opencv` datatype `cv::FeatureDetector`. We create an object of this type and pass to the constructor a number which represents the threshold of the determinant of the Hessian. This means that during the extraction of the keypoints, only the features, with the determinant of the hessian higher than the threshold, are selected. The higher we set the threshold, the less keypoints we will have. The quality of the matching algorithm is directly affected by the number of keypoints extracted. If we have a lot of keypoints the match is going to be finer. However if we filter these keypoints and use only a part of them, we might miss the logo in the image.

For example, in figure 5.2, we have 2 identical images except for the fact that in the first one the keypoints are extracted with a threshold of 500, whereas, in the second one they are extracted with a threshold of 5000. As we can notice, the first image has more keypoints than the second one. Also, after increasing the threshold to 5000, the keypoints are still visible on the image but not on the logo itself which makes it impossible to recognize it.



(a) Coca-cola commercial with a threshold of 500 (b) Coca-cola commercial with a threshold of 5000

Figure 5.2: Coca-cola commercial with different determinant of hessian threshold. We notice that when we increase the threshold, we get less keypoints and therefore we miss the logo in the image.

If we were to summarize what has just been said, the quality that changes the running time of our algorithm is the threshold of the determinant of the hessian. We need to run some tests to learn the behavior of the execution time depending on the quality. And more importantly, we need to set some fixed qualities and know their average running time as we can see in tables 5.1, 5.2 and 5.3.

Table 5.1: Table representing the average execution time (in ms) of the matching algorithm with different threshold values and decreasing the number of keypoints.

Threshold	Div	Keypoints	Average Execution Time	Error(%)
300	1	648	133	15
	2	324	102	16
	3	216	86	16
500	1	513	116	16
	2	256	86	16
	3	171	82	16
750	1	415	106	17
	2	207	81	16
	3	138	74	16
1000	1	338	95	16
	2	169	75	16
	3	112	70	16
2000	1	194	72	16
	2	97	64	16
	3	64	61	17
5000	1	89	62	16
	2	44	57	20
	3	29	not enough matches	

Table 5.2: Table representing the average execution time (in ms) of the matching algorithm with different threshold values and decreasing the number of keypoints.

Threshold	Div	Keypoints	Average Execution Time	Error(%)
300	1	380	99	12.2
	2	190	81	11.9
	3	126	70	11.9
1000	1	320	87	12
	2	160	68	11.8
	3	106	61	11.9
5000	1	210	67	11.8
	2	105	56	11.9
	3	70	53	11.9

Table 5.3: Table representing the average execution time (in ms) of the matching algorithm with different threshold values and decreasing the number of keypoints.

Threshold	Div	Keypoints	Average Execution Time	Error(%)
300	1	964	138	18
	2	482	93	21
	3	322	88	24
1000	1	552	111	19
	2	276	71	21
	3	184	66	23
5000	1	162	55	28
	2	81	42	28
	3	54	no result	

We chose 3 quality levels with their corresponding average time. These qualities are described in table 5.4.

Table 5.4: Table describing the different quality values chosen for our algorithm.

Quality Level	Quality Value 1	Quality Value 2	Quality time
$q_1$	300	3	85
$q_2$	1000	2	70
$q_3$	5000	1	55

The table is composed of four entries. The quality level tells us the rank of the quality that we are using. The quality level ‘1’ stands for the best quality which implies the slowest running time and ‘3’ is the worst quality, however it speeds up the program. The quality time is the average running time of the algorithm using a certain quality. Then, we have two quality values. In fact, our quality depends on two factors. The first one is the threshold of the determinant of the hessian, that we previously discussed. The second one, “quality value 2”, also decreases the number of keypoints used. However, it does so after extracting them. This quality can take three values  $\{1, 2, 3\}$  which represents the number by which we are going to divide the number of keypoints previously extracted.

The last entry of the table is the percentage of error produced by this algorithm. The error function calculates the destination points (in the frame) of every match using the homography matrix computed. Then, it subtracts the calculated coordinates from the original ones and squares the results. This process is done for every match and the results are summed. The error returned is the square root of the result.

### 5.3 Algorithm

Suppose that we want to match our frame to  $m$  logos. This means that we have to repeat our algorithm  $m$  times for the same frame. However our deadline is fixed regardless of the number of logos that we have. In order to stay within the

deadline, knowing that we have to match the image to all the logos, we use the QoS system which is composed of the controller and the parameterized system. The controller, composed of the quality manager, is responsible for determining the most suitable quality for running our algorithm in the following cycle, using the running time of the previous cycle. The quality chosen should be the best quality such that we can still run all the remaining cycles with the worst quality without crossing the deadline. The parameterized system, in our case, is the successive calls of the matching algorithm of the current frame with the different logos, as we can see in figure 5.3.

First, we set the deadline  $D = 198$  ms which represents the available time to execute the matching between a frame and the logos. In the first cycle, the deadline is still  $D$ . The parameterized system sends  $D$  to the controller. The quality manager returns the best quality that can be used if all the following  $m - 1$  cycles can still be the lowest executed with the lowest quality and not cross the deadline  $D$ .

A cycle  $k$ , is the  $k^{th}$  call of the matching algorithm with logo  $k$ . Before executing cycle  $k$ , the parameterized system passes the execution time of the previous cycle  $t_{k-1}$  to the controller and waits for the quality chosen to be used in cycle  $k$ .

Once the controller receives  $t_{k-1}$ , it calculates the new deadline  $d_k$  which represents the remaining time for executing the  $m - k$  cycles left.

$$d_k = d_{k-1} - t_{k-1} \quad (5.1)$$

Then, it calls the quality manager passing the new deadline as argument, see algorithm 11. First, the quality manager calculate the available time  $t_a$  to execute this cycle ( $k$ ), if we decide to run all other cycles ( $m - k$ ) with the worst quality.

$$t_a = d_k - (m - k) \times time(q_3) \quad (5.2)$$

Then, it compares the available time generated  $t_a$  to the time of the best quality

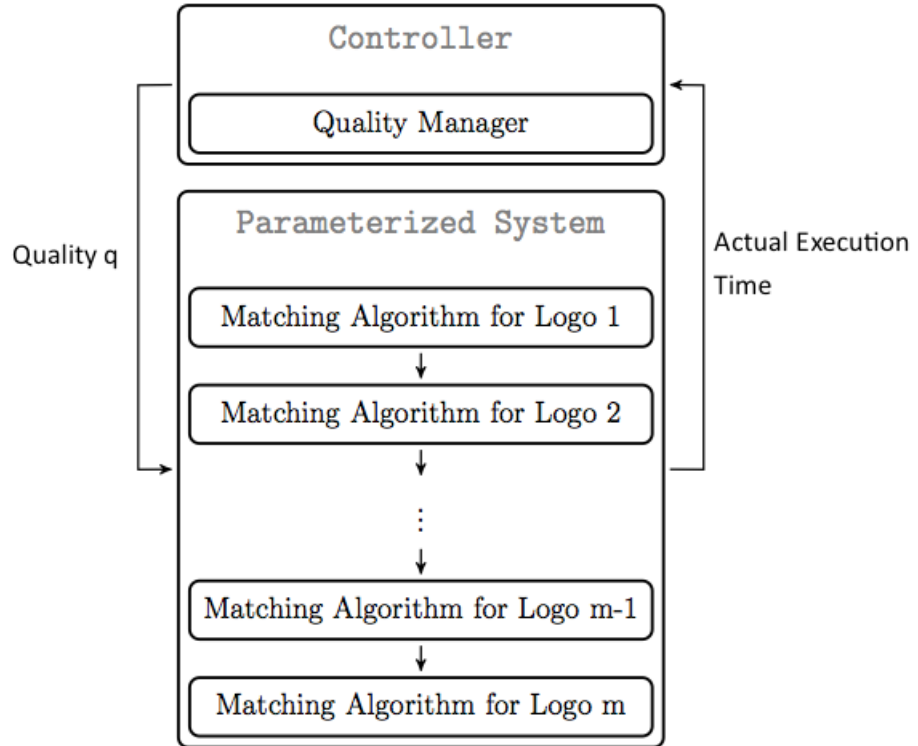


Figure 5.3: Controller architecture

$time(q_1)$ . If  $t_a \geq time(q_1)$  then this means that there is enough time to run our algorithm with this quality, so  $q_1$  is chosen and passed to the parameterized system. If  $t_a \leq time(q_1)$  then we elimination this option and compare the available time to the following quality. This process is continued until a quality is chosen. If the time  $t_a$  is less than the worst quality average time, this means there is not enough time to run this cycle with any available qualities. However, we choose the worst quality, because it is important to run our program even if this means that we cross our deadline.

<p><b>Input</b> : Deadline <math>d</math> and Remaining number of logos <math>m-k</math></p> <p><b>Output</b>: Quality <math>q</math> for cycle <math>k</math></p> <pre> 1 <math>t_a \leftarrow d - (m - k) \times q_3.time;</math> 2 <b>if</b> <math>t_a \geq q_1.time</math> <b>then</b> 3     <math>q_1</math> is selected; 4     exit; 5 <b>else if</b> <math>t_a \geq q_2.time</math> <b>then</b> 6     <math>q_2</math> is selected; 7     exit; 8 <b>else</b> 9     <math>q_3</math> is selected; 10    exit; 11 <b>end</b> </pre>
---

**Algorithm 3:** Quality Manager

Algorithm 10 first loads the database of logos, extracts their respective keypoints and descriptors and constructs the flannIndex for every logo. Then, it captures every six frames of the video and calls algorithm 12 which is responsible for matching a frame to the set of  $m$  logos. For every logo algorithm 12 calls the quality manager function (algorithm 11) passing the new deadline and the number of remaining logos as parameters and expects the quality to be used in return. Then, it calls the matching function to match the frame to a logo with the chosen quality. We calculate the executing time of this function and we deduct it from the deadline. This process is done for all the logos.



```

Input : The frame, logos, number of logos m
Output: The frame with the detected logos
1 current_deadline  $\leftarrow$  deadline;
2 current_numOfLogo  $\leftarrow$  m;
3 while current_numOfLogo  $\neq$  0 do
4   q  $\leftarrow$  getQuality(current_deadline, current_numOfLogo);
5   index  $\leftarrow$  m - current_numOfLogo;
6   startTime;
7   matchImage(frame, logo[index], flannIndex[index], q);
8   endTime;
9   time  $\leftarrow$  endTime - startTime;
10  current_deadline = current_deadline - time;
11  current_numOfLogo--;
12 end

```

**Algorithm 4:** Decides which quality to use and call the matching algorithm with the quality chosen.

<p><b>Input</b> : Video and a logo</p> <p><b>Output</b>: The video with the logo detected and found.</p> <pre> <b>1</b> for <math>i \leftarrow 0</math> to <math>m</math> do <b>2</b>     logo_keypoints[<math>i</math>] <math>\leftarrow</math> extractKeypoints(logo[<math>i</math>]); <b>3</b>     logo_descriptors[<math>i</math>] <math>\leftarrow</math> extractDescriptors(logo[<math>i</math>]); <b>4</b>     flannIndex[<math>i</math>] <math>\leftarrow</math> flannIndex.build(logo_descriptors[<math>i</math>],         cv::flann::KDTreeIndexParams(), cvflann::FLANN_DIST_EUCLIDEAN); <b>5</b> end <b>6</b> while <i>not finished</i> do <b>7</b>     frame <math>\leftarrow</math> video.getframe(); <b>8</b>     matchImageToLogos(frame, logo, m); <b>9</b>     display(frame); <b>10</b> end </pre>
---

**Algorithm 5:** Matching Algorithm for a video

## 5.4 Experimental Results

In order to check if the quality manager is selecting different qualities for the different logos while respecting the deadline, we test our algorithm first, by using a dataset consisting of 3 logos and second, using a dataset of 6 logos.

### *Dataset of 3 logos*

We match every 6 frames of a video to 3 logos and we count the number of occurrences of every quality chosen by the quality manager. The results are shown in figure 5.4. As we can see, the quality manager chose the first quality  $q_1$  (best quality) most of the time.

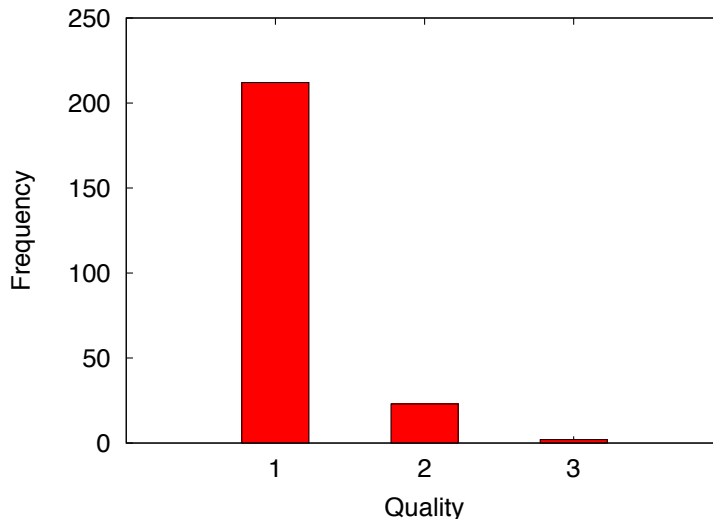


Figure 5.4: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 3 logos.

The goal is to test the performance of the quality manager. Therefore, we compare the execution times of our program using the quality manager to a program using a fixed average quality  $q_2$ . In both cases, the deadline hasn't been crossed, however, the program using the quality manager takes more time to execute than the one using the fixed quality  $q_2$ . This result was expected since the quality manager is choosing the best quality most of the time. This clearly shows that the quality manager is maximizing the quality of the results while respecting the deadline.

### ***Dataset of 6 logos***

We match every 6 frames of a video to 6 logos and we count the number of occurrences of every quality chosen by the quality manager. The results are shown in figure 5.5. As we can see, the quality manager chose the worst quality  $q_3$  most of the time.

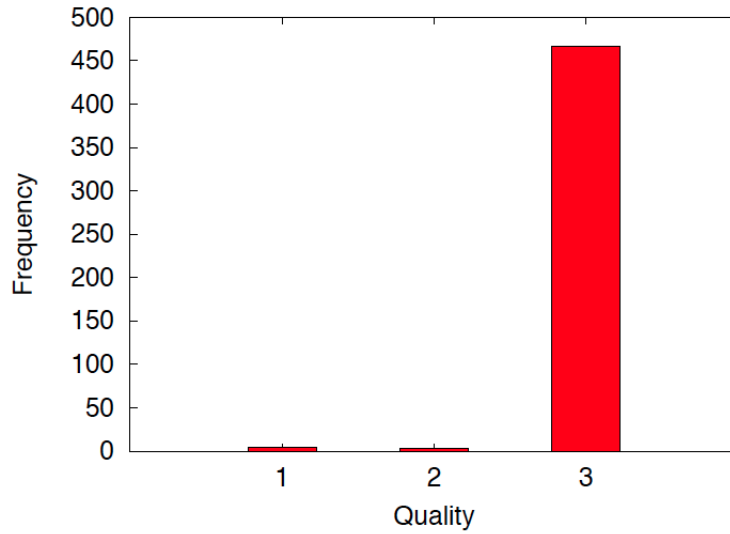


Figure 5.5: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 6 logos.

We test our algorithm based on the quality manager by trying to find 6 logos in the video. We calculate the execution time of matching every frame to the 6 logos, then we compute their average. This is done for the program using the quality manager and it is also done for the program using a fixed quality. We choose this quality to be  $q_2$  which represents the average quality.

We want to compare the execution time of both algorithms. In the first case (program using the quality manager), we didn't cross the deadline; instead we still have 11 ms left.

$$D - ExecutionTime(qualitymanager) = 11 \quad (5.3)$$

Whereas, in the second case (program with the fixed quality), we crossed the deadline by 83 ms.

$$D - ExecutionTime(q_2) = -83 \quad (5.4)$$

These results show the importance of the quality manager. In fact, while using the quality manager, we managed to match every frame to the set of logos without crossing the deadline. The resulting speedup isn't at the cost of performance as we

can see in figure 5.6 and 5.7. The quality manager chose the worst quality ( $q_3$ ) 98% of the time.



Figure 5.6: Screenshot of the resulting video using the quality manager



Figure 5.7: Screenshot of the resulting video using the fixed quality  $q_2$

Figure 5.6 shows the results using the quality manager while figure 5.7 shows the results using the fixed average quality. We can clearly see that the results are similar in both images.

## CHAPTER 6

### PARALLELIZE OUR ALGORITHM USING PTHREADS

#### 6.1 Introduction

The introduction of the notion of quality helped us manage real-time execution. However, if we are skipping 5 frames, this means that our deadline for the execution of one frame is  $D = 198$  ms. If we consider that we are matching our frame to every logo using the worst quality (60 ms), this implies that we can work with at most 3 logos ( $198/60 = 3.3$ ). We would like to be able to match our frame to more than 3 logos. This can be done if we match every 3 logos in parallel.

Parallelism can be done in two ways: threads and processes. A thread is a set of instructions that can be run independently from the operating system. Usually a program being multi-threaded is a program that has several threads that can be run simultaneously. Different threads that are part of the same process share resources. This means that if one thread makes changes to this shared data, changes will be seen by all other threads. It is also known as shared memory parallel programming, whereas, a process is known as non-shared memory parallel programming. Despite what we just stated, a process has its own private memory. Since processes don't have a shared memory as threads, the only way for them to share information is by passing messages. It is also called message passing parallel programming.

Creating and destroying processes are more expensive than creating and destroying threads. They are also found to have a longer life than threads. However, they are not dynamic unlike threads which can be destroyed and created as needed. Since, we are going to create  $n$  threads for every 5 frames of the video, it is most suitable to use threads.

## 6.2 Algorithm

Our main goal is to match our frame to more than 3 logos. For this, we create several threads in parallel, each of which matches our frame to 3 logos. Suppose that we have  $n$  threads and  $m$  logos. This means that every thread, will have  $\frac{n}{m}$  logos to match to the main frame.

```
Input : Video, logos and number of threads  $n$ .  
Output: The video with the logos detected and found.  
1 for  $i \leftarrow 0$  to  $m$  do  
2 |   logo_keypoints[ $i$ ]  $\leftarrow$  extractKeypoints(logo[ $i$ ]);  
3 |   logo_descriptors[ $i$ ]  $\leftarrow$  extractDescriptors(logo[ $i$ ]);  
4 |    $flannIndex[ $i$ ] \leftarrow flannIndex.build(logo_descriptors[ $i$ ],$   
|   cv::flann::KDTreeIndexParams(), cvflann::FLANN_DIST_EUCLIDEAN);  
5 end  
6 while not finished do  
7 |   frame  $\leftarrow$  video.getframe();  
8 |   for  $t \leftarrow 0$  to  $n$  do  
9 | |   thread_parameter[ $t$ ].start  $\leftarrow t \times m/n$ ;  
10 | |   thread_parameter[ $t$ ].img  $\leftarrow$  &frame;  
11 | |   pthread_create(&tid[ $t$ ], NULL, FindLogo, &thread_parameter[ $t$ ]);  
12 |   end  
13 |   for  $t \leftarrow 0$  to  $n$  do  
14 | |   pthread_join(tid[ $t$ ], NULL);  
15 |   end  
16 |   display(frame);  
17 end
```

**Algorithm 6:** Matching Algorithm for a video

After loading the frame, we usually used to call the function FindLogo() which matches the frame to all  $m$  logos respecting the deadline  $D$  as possible. However,

we introduced threads to divide the work to be done by  $n$ , in order not to cross the deadline. In Algorithm 6, we create  $n$  threads, as we can see on line 11, by calling the following function `pthread_create()`. It takes 4 arguments as parameters:

1. The first parameter is a pointer to the thread id. Every thread should have an id in order to be able to distinguish it from other threads.
2. The second one is the thread's attributes.
3. The third one is the name of the function that will be executed by the thread. In our case, it is `FindLogo`.
4. The fourth one is the arguments that the function takes. In this case, instead of passing all the logos to the function we are only passing the needed logos. We have  $n$  threads and  $m$  logos, this means that we have  $m/n$  logos per thread. So, thread 1 will match the frame to the first  $m/n$  logos, thread 2 will match the frame to the following  $m/n$  logos,  $\dots$ , thread  $n$  will match the frame to the last  $m/n$  logos.

We call `pthread_join()` in order to make sure that all the threads are completed before displaying the resulting image.



<p><b>Input</b> : The frame, logos, number of logos <math>m</math></p> <p><b>Output</b>: The frame with the detected logos</p> <pre> 1 current_deadline <math>\leftarrow</math> deadline; 2 current_numOfLogo <math>\leftarrow m/n</math>; 3 while <i>current_numOfLogo</i> <math>\neq</math> 0 do 4   q <math>\leftarrow</math> getQuality(current_deadline, current_numOfLogo); 5   index <math>\leftarrow</math> parameter.start + <math>m/n</math> - current_numOfLogo; 6   frame <math>\leftarrow</math> parameter.img; 7   startTime; 8   matchImage(frame, logo[index], flannIndex[index], q); 9   endTime; 10  time <math>\leftarrow</math> endTime - startTime; 11  current_deadline = current_deadline - time; 12  current_numOfLogo--; 13 end </pre>
---

**Algorithm 7:** Decides which quality to use and call the matching algorithm with the quality chosen (FindLogo).

### 6.3 Experimental Results

We run our algorithm several times using one to four threads as shown in table 6.1. We tested our algorithm first, by using 3 logos and we varied the number of threads. It took 150 ms on average to match a frame to 3 logos using one thread. As we described earlier, the quality manager used the best quality most of the time, see figure 6.1. The results are similar when matching a frame to 3 logos using 2, 3 or 4 threads.

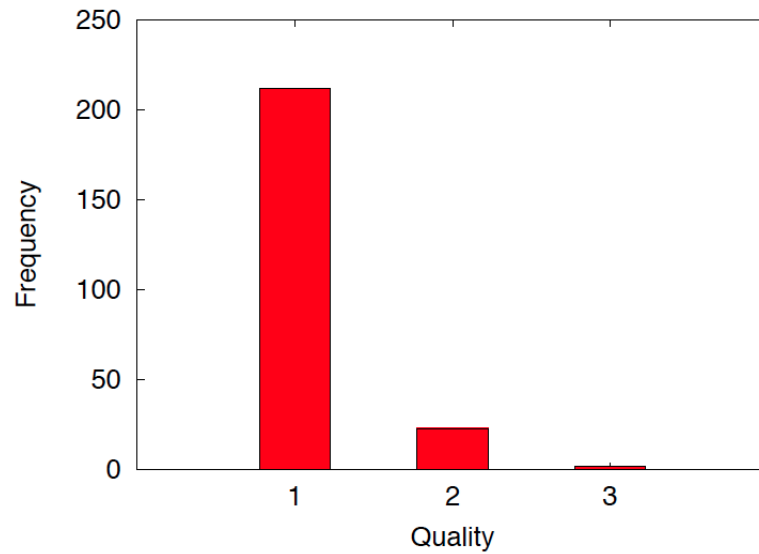


Figure 6.1: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 3 logos using one thread.

Table 6.1: Table representing the average execution time of a frame matched with the specified number of logos using 1, 2, 3 or 4 threads.

Number of Logos	1 Thread	2 Threads	3 Threads	4 Threads
3	150	169	160	160
6	187	165	187	190
9	272	297	194	200
12	362	354	354	236

Second, we tested our algorithm by matching a frame to 6 logos while varying the number of threads. As we can see in figure 6.2, the quality manager chose the best quality most of the time while matching a frame to 6 logos with two threads.

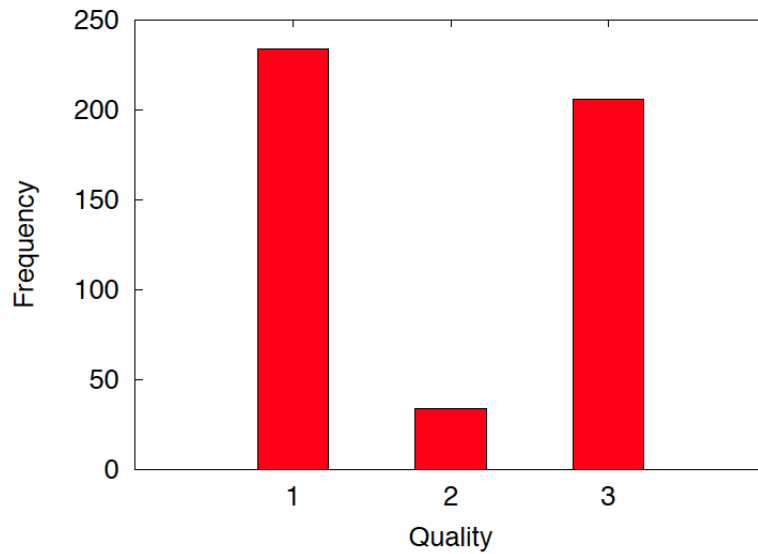


Figure 6.2: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 6 logos using 2 threads.

It takes 165 ms to execute which is less than the execution time of matching 6 logos with one thread (187 ms). We get this speedup with a gain in quality. In fact, in figure 6.3 we can see that the quality manager only used the worst quality to match the 6 logos, whereas using two threads we notice that it is using the best quality most of the time.

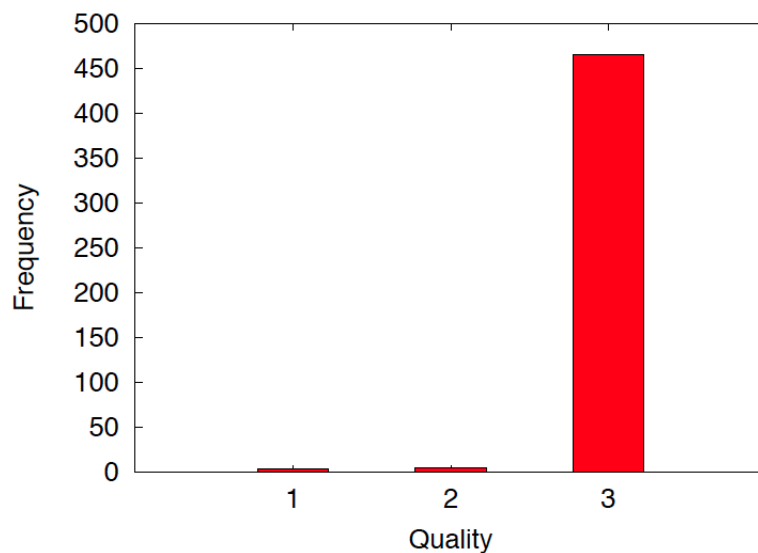


Figure 6.3: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 6 logos using 1 thread.

We get similar results when matching 9 (figure 6.4) and 12 (figure 6.5) logos to

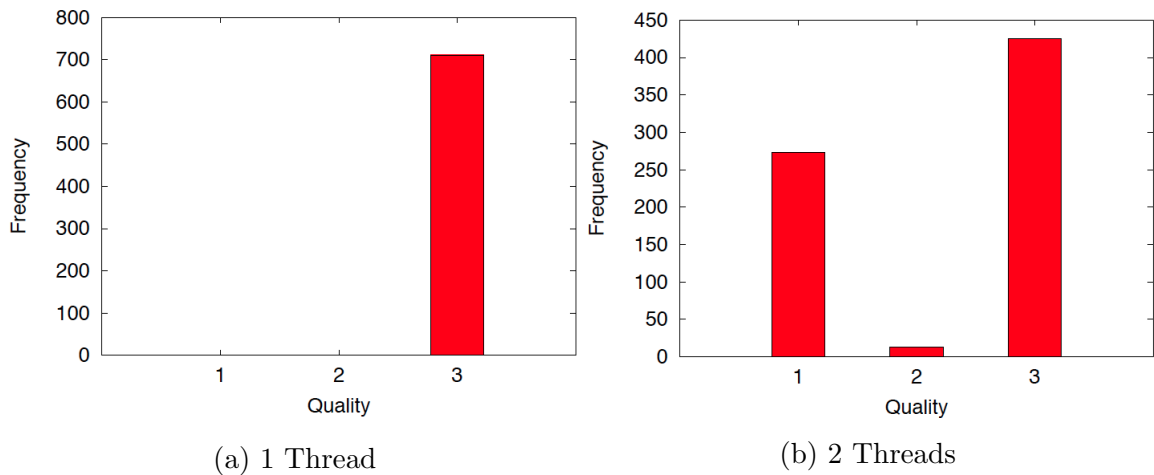
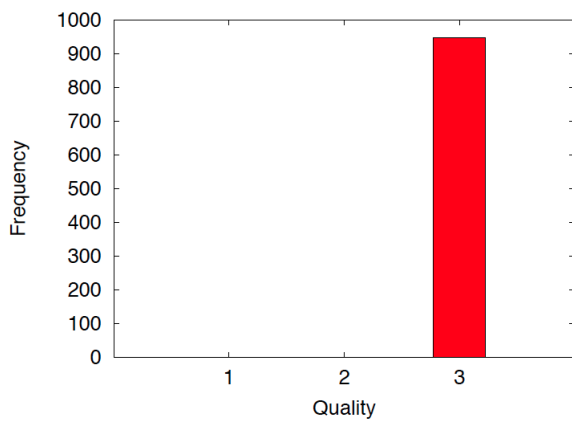
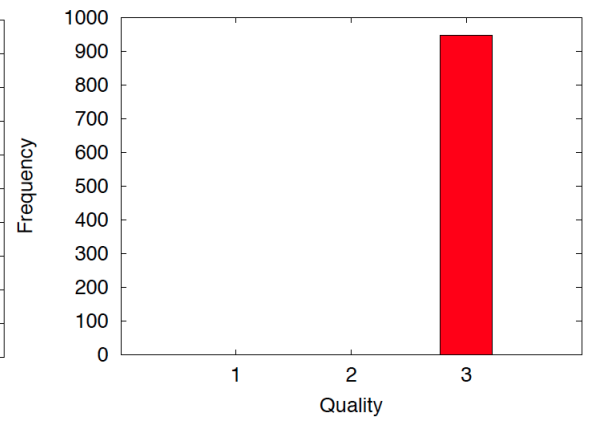


Figure 6.4: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 9 logos.

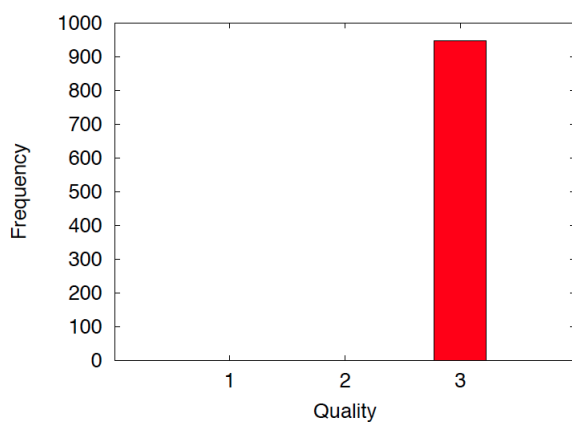
the video. The results have been made using a computer that integrates Intel Core i5-2410M @ 2.29 GHz (4 Cores). This computer can run up to 4 threads in parallel, this means that we can match a video up to 12 logos without exceeding the deadline. Having more cores would lead us to match a video to a bigger set of logos.



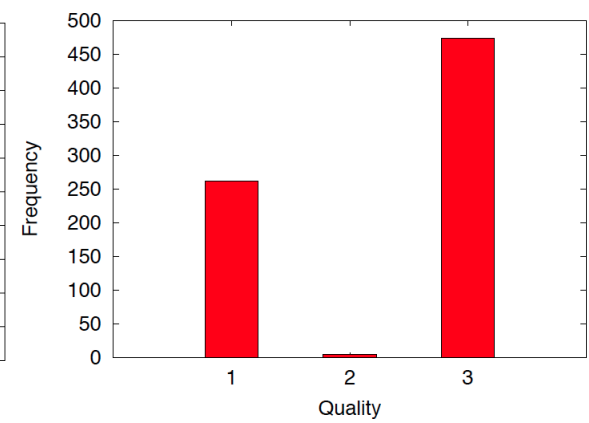
(a) 1 Thread



(b) 2 Threads



(c) 3 Threads



(d) 4 Threads

Figure 6.5: Number of occurrences of every quality chosen by the quality manager while matching a video to a set of 12 logos.

## CHAPTER 7

### CONCLUSION

The goal of this thesis was to recognize and localize logos in a video in real-time. However, this task was hard to accomplish since every frame has only 33 ms to be matched to the set of logos. Because successive frames are mostly similar, we extended this deadline by six. Thus, we apply the matching algorithm every six frames. With the use of the controller and the introduction of different qualities, we managed to control the execution time of the algorithm. The most important part was to respect the deadline set for the matching process and to maximize the use of the time budget (i.e. use the best quality possible within the time constraints). Using the quality manager, we were able to match a video to 3 logos, on average, in real time. Finally, we parallelized our code in order to be able to match our video to a larger set of logos.

The tests have been done on a computer which integrates Intel Core i5-2410M @ 2.29 GHz (4 cores). This means that it can only run 4 threads in parallel. Therefore, we can find up to 12 logos in a video in real-time using this computer. With more powerful machines, we can match our video to a larger set of logos (in real-time), using more threads.

#### **Future Work**

Every brand has a set of nuances of the same logo. Sometimes the colors are different if the product is “diet”, or the colors are switched depending on the background color. However, the matching algorithm used does not take these cases into consideration. Therefore, a logo with these nuances will not be found. As future work, we could create an index for each logo based on the set of nuances of this logo.

So instead of matching our image to one variance of the logo we are matching it to a set of logos.

In order to enlarge our dataset of logos, we could also create several indices (one for every logo) and run the FLANN index matcher in parallel on these trees. This would also require a more powerful machine to extend the number of threads that can be run in parallel.

We implemented our quality manager using a safe scenario algorithm that favors the first logos. For future work we could use a function that will distribute the qualities equitably.

This program can be easily integrated on Android and IOS devices. Moreover, it can be added as a YouTube plugin for advertisement purposes.

## REFERENCES

- October 2006. URL <http://www.opencv.org>.
- Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features. *Elsevier*, September 2008.
- Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. *RTSS*, pages 286–295, 1998.
- Jacques Combaz, Jean-Claude Fernandez, Thierry Lepley, and Joseph Sifakis. Qos control for optimality and safety. *Proceedings of the 5th Conference on Embedded Software*, September 2005b.
- Jacques Combaz, Jean-Claude Fernandez, Joseph Sifakis, and Loic Strus. Symbolic quality control for multimedia applications. October 2008.
- M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, pages 381–395, 1981.
- Google. Google glass. URL <http://www.google.com/glass/start/>.
- Martin Herdina, Andy Gstoll, and Martin Lechner. Wikitude. URL <http://www.wikitude.com/>.
- Gilad Koren and Dennis Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. *Technical Report*, TR1996-715, 1996.
- T. Lindeberg. Feature detection with automatic scale selection. *IJCV*, pages 79–116, 1998.
- David G. Lowe. Object recognition from local scale-invariant features. *International Conference on Computer Vision*, pages 1150–1157, September 1999.



- Chenyang Lu, John A. Stankovic, Gang Tao, and Sang H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23:85–88, 2002.
- J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. *BMVC*, pages 384–393, 2002.
- K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. *ECCV*, pages 128–142, 2002.
- K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *IJCV*, (63-86), 2004.
- K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *PAMI*, pages 1615–1630, 2005.
- Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *International Conference on Computer Vision Theory and Application VISSAPP'09*, pages 331–340, 2009.
- Marius Muja and David G. Lowe. Fast matching of binary features. *Computer and Robot Vision (CRV)*, pages 404–410, 2012.
- Edouard Oyallon and Julien Rabin. An analysis and implementation of the surf method, and its comparison to sift. *Image Processing On Line*, February 2013.
- P. M. Panchal, S. R. Panchal, and S. K. Shah. A comparison of sift and surf. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2), April 2013.
- Quintin and Maarten. Layar, Summer 2009. URL <https://www.layar.com/>.

- Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. A resource allocation model for qos management. *EEE Real-Time Systems Symposium*, pages 298–307, December 1997.
- R.I.Davis, K.W.Tindell, and A.Burns. Scheduling slack time in fixed priority preemptive systems. *Proceeding of the IEEE Real-Time Systems Symposium*, pages 222–231, 1993.
- S. Se, H.K. Ng, P. Jasiobedzki, and T.J. Moyung. Vision based modeling and localization for planetary exploration rovers. *Proceedings of International Astronautical Congress*, 2004.
- T. Tuytelaars and L. Van Gool. Wide baseline stereo based on local, affinely invariant regions. *BMVC*, pages 412–422, 2000.
- Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *CVRP*, pages 511–518, 2001.
- Y. Wu, M. El Choubassi, and I. Kozintsev. Augmenting 3d urban environment using mobile devices. *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, October 2011.
- Clemens C. Wust, Liesbeth Steffens, Wim F.J. Verhaegh, Reinder J. Bril, and Christian Hentschel. Qos control strategies for high-quality video processing. *Euromicro Conference on Real-Time Systems*, pages 3–12, 2004.