Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Centro de Investigación y Estudios de Posgrado

Parallelization of the Honeybee Search Algorithm for Video Tracking

T E S I S

Que para obtener el grado de:

Maestro en Ingeniería de la Computación

Presenta:
Oscar Ernesto Pérez Cham

Asesor:

Dr. Cesar Augusto Puente Montejano

San Luis Potosí, S. L. P. Agosto de 2017

UASLP
Universidad Autónoma
de San Luis Potosí

15 de junio de 2017

ING. OSCAR ERNESTO PÉREZ CHAM
P R E S E N T E.

En atención a su solicitud de Temario, presentada por el **Dr. César Augusto Puente Montejano,** Asesor de la Tesis que desarrollará Usted, con el objeto de obtener el Grado de **Maestro en Ingeniería de la Computación**, me es grato comunicarle que en la Sesión del H. Consejo Técnico Consultivo celebrada el día 15 de junio del presente, fue aprobado el Temario propuesto:

TEMARIO:

"Paralelización del Algoritmo de Búsqueda de las Abejas para el seguimiento de objetos en video"

Introducción.
1. Trabajo Previo en Algoritmos Inspirados en Abejas y Seguimiento de Objetos en Video.
2. Marco Teórico sobre la Paralelización de Algoritmos de Inteligencia de Enjambre.
3. Seguimiento de Objetos en Video Usando el Algoritmo de Búsqueda de las Abejas Paralelo.
4. Experimentos Utilizando una unidad de Procesamiento Gráfico.
5. Conclusiones.

Referencias.

"MODOS ET CUNCTARUM RERUM MENSURAS AUDEBO"

A T E N T A M E N T E

M. I. JORGE ALBERTO PÉREZ GONZÁLEZ
DIRECTOR.

UNIVERSIDAD AUTÓNOMA
DE SAN LUIS POTOSI
FACULTAD DE INGENIERIA
DIRECCION

Copia. Archivo.
*etn.

FACULTAD DE
INGENIERÍA

Av. Manuel Nava 8
Zona Universitaria • CP 78290
San Luis Potosí, S.L.P.
tel. (444) 826 2330 al39
fax (444) 826 2336
www.uaslp.mx

Dedicado a mis padres.

Acknowledgments

Thanks to Dr. Cesar Augusto Puente Montejano for his constant guidance, interest and collaboration during the development this work.

To Raymundo Antonio González Grimaldo and Dr. Juan Carlos Cuevas Tello for sharing their knowledge about Parallel Computing.

To CONACYT (Consejo Nacional de Ciencia y Tecnología) Ciencia Básica for supporting this work through scholarship (Project No. 177041).

To my family and friends for all their help and concern.

Resumen

El seguimiento de objetos en video es uno de los muchos problemas en el campo de la Visión Computacional; es un componente básico para muchos y más complejos sistemas de visión que son útiles en varias aplicaciones del mundo real en áreas como investigación médica, vigilancia, robótica, telecolaboración, etc. Un algoritmo de seguimiento de objetos en video intenta rastrear un objeto de interés a través de los cuadros de cierta secuencia de video. Esta tesis estudia los efectos de llevar a cabo el seguimiento de objetos en video con la ayuda del Algoritmo de Búsqueda de las Abejas, un algoritmo de Inteligencia de Enjambre inspirado en la búsqueda de fuentes de alimento que realizan las abejas; y Unidades de Procesamiento Gráfico, que son un ejemplo de una tecnología de Computo Paralelo diseñada específicamente para operaciones de renderización gráfica. Los resultados prueban que es posible paralelizar el Algoritmo de Búsqueda de las Abejas y usarlo para el seguimiento de objetos en video. En comparación con una versión paralela del mismo algoritmo de seguimiento de objetos en video, la adición del Algoritmo de Búsqueda de las Abejas ayuda a proveer un tiempo más estable para entregar resultados, haciéndolos menos dependientes del tamaño del video, y sin causar efectos negativos notables en la precisión de los resultados.

Abstract

Video tracking is one of the many problems in the field of Computer Vision; it is a basic component for many and more complex vision systems that are useful for several real world applications in the areas of medical research, surveillance, robotics, tele-collaboration, etc. A video tracking algorithm tries to follow an object of interest trough the frames of a given video sequence. This thesis studies the effects of performing video tracking aided by the Honeybee Search Algorithm, a Swarm Intelligence (SI) algorithm that is inspired in the foraging behavior of honeybees; and Graphics Processing Units (GPUs), which are an example of a Parallel Computing technology designed specifically for graphic rendering operations. The results prove that it is possible to parallelize the Honeybee Search Algorithm and use it for video tracking. In comparison with a parallel version of the same video tracking algorithm, the addition of the Honeybee Search Algorithm helps to provide a more stable time to deliver results, making them less dependent on the size of the specific video, and without causing notable negative effects in the accuracy of the results.

Contenidos

Contents

List of Figures

List of Tables

Introduction

Since the beginnings of computer science there has always been interest in allowing the computer to think and display some kind of intelligent behavior, if such thing is possible (Turing, 1950). Decades later, the problems of Artificial Intelligence remain unsolved, but research of this topic has produced many breakthroughs that have become pervasive in daily life. This can be verified by simply listing some of the applications of Computer Vision, a smaller division of Artificial Intelligence that deals specifically with the problem of allowing the computer to see and understand what it sees (Bradski & Kaehler, 2008). Thanks to Computer Vision, cameras can automatically calibrate (Li & Lavest, 1996), there is automatized surveillance (Tian et al., 2008), humans can communicate with computers using body gestures (Zhang, 2012), production lines can be inspected by computers (Ntuen, Park, & Kim, 1989), there are even vehicles that can drive themselves (Nothdurft et al., 2011).

One of the problems that Computer Vision faces is the problem of following certain object that is captured in video, this problem is commonly known to researchers as video tracking (Trucco & Plakas, 2006). The mentioned problem is only a basic step, generally useful for many of the specific applications of Computer Vision. Many different approaches have been suggested in an

attempt to solve the problem of video tracking but no definitive answer has been found. Moreover, there is also the problem of making video tracking faster but reliable for it to be useful in applications where real-time is the most relevant requirement (Galoogahi, Fagg, Huang, Ramanan, & Lucey, 2017).

Computer Vision is not left alone suffering with variables such as speed and time, its a fairly generalized problem for all the fields of computer science (Crescenzi & Kann, 1997). This is why Parallel Computing technologies were born and continue to thrive as useful tools whenever heavy volumes of data have to be analyzed (Foster, 1995). Artificial Intelligence has also found inspiration in how biological beings provide time efficient solutions to the same problems, giving foundations for areas such as Evolutionary Computing and Swarm Intelligence (Bitam, Batouche, & Talbi, 2010). The efforts to adopt those tools and use them for Computer Vision and many other problems of computer science have already started and proven to be a productive research topic (Tan & Ding, 2016).

The Matching Problem

Digital images that can be displayed by computers provide data in a very crude format. These images are made by pixels, small units that could be seen as composed of two discrete pieces of data: position and color. A computer can only perceive an object in a specific image as a group of pixels, but that same object is a different set of pixels in another image. It is expected that there is some similarity between those groups of pixels, especially if

both images come from the same camera at different but close points in time. Trying to establish that the first group of pixels represents the same object as the second group of pixels is commonly known as the matching problem (Smeulders et al., 2014; Trucco & Plakas, 2006). Video tracking can be reduced to trying to solve the matching problem many times, once per image.

Zero Mean Normalized Cross-Correlation, for example, measures the similarity of each pixel in the first group against a given pixel in the second group (Di Stefano, Mattoccia, & Tombari, 2005). Once the similarity of every pair of pixels is measured, the overall similarity is calculated by statistical methods. Needless to say, comparing each pair of pixels and then going through the results to summarize them statistically is a heavy task involving a big number of operations. And that has to be done once for every suspicious spot in the image. The matching problem can then be seen as a search problem, where many options have to be reviewed in search for the correct answer.

Population Based Optimization

Since the matching problem (the basis of video tracking) can be seen as a search problem, it is possible to use Population Based Optimization approaches such as Evolutionary Algorithms and Swarm Intelligence to optimize the way in which the search is performed. Both Evolutionary Algorithms and Swarm Intelligence are based on the idea of performing smart searches that do not review every single possible option. Only a small portion of the possible

answers are actually evaluated, depending on where the best solutions are found, the search process pays more attention to answers that are close to those spots.

One of the algorithms that have emerged from this trend is the Honeybee Search Algorithm, which is based on how honeybees search for food (Olague & Puente, 2006a). First a quick search with few resources is performed to get information about the general location of the best solutions. That information is later used to assign search resources for a second more extensive search that pays more attention to the places where the best solutions were previously found.

There have been several attempts to parallelize Swarm Intelligence algorithms. Since these algorithms are commonly based on the coordination procedures of many individuals, it is only natural to try to emulate how those individuals perform their work simultaneously.

Parallel Computing for Swarm Intelligence

Finding the best way to implement parallel versions of Swarm Intelligence Algorithms is currently drawing much attention from researchers. More specifically, Graphic Processing Units have been of great interest to parallelize Swarm Intelligence algorithms because of their availability in comparison with other technologies such as clusters and grids (Tan & Ding, 2016). This is because Graphic Processing Units have evolved to be available commodity hardware. Clusters and grids, on the other hand, can be more expensive as several common computers and networking supplies are needed to assemble

them; the difference between both of them is that while clusters have a rigid centralized coordination approach; grids are built to work as collaborating peers (Prabhu, 2008). Chapter 2 provides more information on the subject of how to implement parallel Swarm Intelligence algorithms with Graphics Processing Units.

Objectives of This Thesis

The general objective of this thesis is:

> *To design and implement a methodology to parallelize the Honeybee Search Algorithm and apply it to the problem of video tracking using a Graphics Processing Unit.*

The specific objectives are listed below:

- To understand and define video tracking and different methods that have been proposed to tackle this problem.
- To know the way in which Evolutionary Computing and Swarm Intelligence have been used to deal with similar problems.
- To review how the Honeybee Search Algorithm works and how it could be used for video tracking.
- To research the different Parallel Computing tools that are available.
- To investigate how Graphics Processing Units are used and how these could be used to develop a parallel version of the Honeybee Search Algorithm.

- To implement a computer program that performs video tracking and uses the Parallel Honeybee Search Algorithm running in a Graphics Processing Unit.

- To evaluate and compare the computer program that is developed against other algorithms of the state-of-the-art and against the parallel version of the same video tracking method that is selected.

Organization of This Thesis

Chapter 1 will define the problem of video tracking and provide an overview of different methods that have already been proposed with the intention of solving it. The same chapter will also introduce the concepts of Evolutionary Computing and Swarm Intelligence. The approaches that are based on the behavior of honeybees will be discussed with greater detail. Parallel Computing technologies will be discussed in Chapter 2; the chapter is also concerned with proving a framework to adapt Swarm Intelligence for Parallel Computing. Graphics Processing Units are introduced as a promising alternative to deal with problems that deal with images. To continue, Chapter 3 will describe the methodology that was used to adapt a video tracking function, along with certain algorithm that uses certain Swarm Intelligence, for Parallel Computing. The experiments and evaluation of the program that is implemented will be narrated in Chapter 4.

Chapter 1

Previous Work on Algorithms Inspired by Honeybees and Video Tracking

The present chapter serves as a theoretical background that exposes the basics to understand following chapters, where the actual contributions of this thesis are described in detail. This theoretical background is mainly concerned with two broad subjects: the problem of video tracking, and algorithms that are inspired by honeybees.

The part of this chapter that discusses video tracking (section 1.1) will provide a definition of the problem itself, and explain different types of algorithms that have been proposed in an attempt to solve it, at least to certain degree. There are two algorithms of special interest for this thesis: Zero Mean Normalized Cross-Correlation (ZNCC) and the Sobel filter. As will be described in following chapters, both methods were used in the experiments and methodology of this thesis; this is why these will receive a

greater attention.

The other part of this chapter (section 1.2) will discuss how and why computer science takes inspiration from biological subjects such as evolution, and the behavior of honeybees and other social insects. Introductions will also be provided for relevant concepts such as Swarm Intelligence (SI), Evolutionary Algorithms (EAs), and other important points. This part of the chapter will also describe the Honeybee Search Algorithm, which is one of the main pillars of this thesis.

1.1 Video Tracking

Video tracking is one of the many problems in the field of Computer Vision, how to solve this problem in definitive remains an open question for computer scientists. As any problem of Computer Vision, video tracking is an attempt to obtain information from raw visual data in order to make complex decisions based on it (Patnaik & Yang, 2012). This visual data usually proceeds from still two-dimensional images; video itself is simply a sequence of these images that can be helpful when variables such as time and movement are also of interest (Trucco & Verri, 1998). The following section provides a definition for video tracking and talks about its importance for real world applications; it also presents a categorization for several video tracking algorithms; later two techniques of Computer Vision that are of special interest for this research are described in detail: Zero Mean Normalized Cross-Correlation (ZNCC) and the Sobel filter, it is also justified why these techniques were selected.

As defined by Mohanapriya and Mahesh (2017), video tracking is the

problem of finding an object of interest in the image frames of some video sequence. It is important to note that the object in question is already located and marked on the initial frame, meaning that the location of that object has to be provided as input (Figure 1.1). Video tracking is not concerned with how or why that object of interest was initially marked; it is only interested in following it even when the objects of interest change their physical appearance between frames (Smeulders et al., 2014; Trucco & Plakas, 2006).

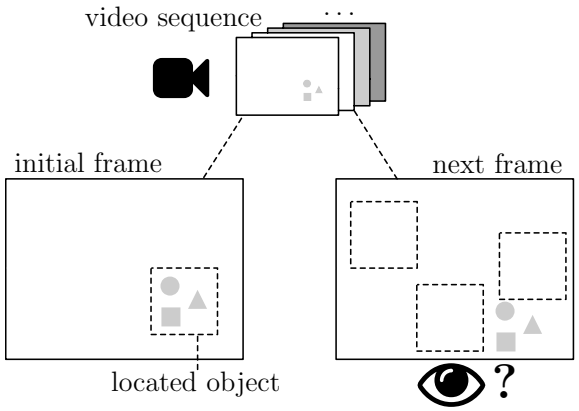

Figure 1.1: In video tracking, the object is already located in the initial frame. The problem is to find it in other consecutive frames despite changes in shape, light, or other variables

The reason why the video tracking problem gets attention is because a lot of complex computer vision systems need it as a basic component. The real world applications of video tracking are too many to list, but Maggio and Cavallaro (2011) have proposed a number of broad categories for the classification of said applications. A specific application of video tracking could belong to one or more of the following categories:

- **Medical applications and biological research:** an example is

breast cancer location and detection with digital image elasto-tomography (Hii, Hann, Chase, & Van Houten, 2006; Van Houten, Kershaw, Lotz, & Chase, 2012); also the analysis of how a person walks to evaluate the condition of bones and joints (Kaufman, Hughes, Morrey, Morrey, & An, 2001); and the automatized observation of Escherichia coli bacteria (Xie, Khan, & Shah, 2008).

- **Surveillance and business intelligence:** applications such as detecting forest fires (Rao, Rao, Duvvuru, Bendalam, & Gemechu, 2016); complex event-based surveillance systems for airport environments (Tian et al., 2008); and obtaining interesting data for retailers such as customer count and effectiveness of merchandising (Senior et al., 2007).

- **Robotics and unmanned vehicles:** an specific case are cars that drive themselves in urban environments (Nothdurft et al., 2011); another one is robots used for automation of industrial manufacturing (Malamas, Petrakis, Zervakis, Petit, & Legat, 2003); and also humanoid robots that assist in people in their homes (Castillo, 2016).

- **Media production and augmented reality:** face motion capture for computer animation, for instance (Maurer, Elagin, Nocera, Steffens, & Neven, 2001); automatic video stabilization (Litvin, Konrad, & Karl, 2003); even adding computer-generated objects in video sequences, either after or during video capture (Hoshino, Yamamoto, & Saito, 2001).

- **Tele-collaboration and interactive gaming:** for example, interaction with games using full body gestures in a natural way (Zhang,

2012); following the gaze of the expositor in a conference to simulate eye contact is also possible (Gemmell, Toyama, Zitnick, Kang, & Seitz, 2000).

- **Art installations and performances:** Sensetable, which is a wireless object tracking platform for tangible user interfaces, can be considered in this category (Patten, Ishii, Hines, & Pangaro, 2001); SwarmArt, that uses tracking of the movements of participants to interact with projected images and illumination installations, can be considered too (Boyd, Hushlak, & Jacob, 2004).

Many of the actual applications of video tracking that were just introduced need results to be provided in real-time, or in other words, almost as fast as cameras are able to generate images. This can be problematic, cameras are capable of producing images very fast, traditionally a common video would have around 30 frames per second, but recently speeds of 240 frames per second are becoming more and more common (Galoogahi et al., 2017). This is why there is a great interest to improve the speed of video tracking algorithms, but not sacrificing the accuracy that has already been obtained. In any case, a video tracking algorithm should be selected based on the requirements, for example, real-time is a must for unmanned vehicles but timely response could be less important for art installations.

1.1.1 Video Tracking Algorithms

Smeulders et al. (2014) makes an important distinction when talking about video tracking algorithms, these can be divided in online trackers and offline

trackers. Offline trackers are the ones that provide results only after analyzing the whole video sequence, which allows them to go through it several times and even backwards if necessary. Online trackers are the opposite, these deliver results one frame at a time which can harm the accuracy. Ultimately, online trackers are the ones that are actually adequate for most applications, because decisions have to be made based on the results as soon as each frame is ready. It should be noted that online trackers do not necessarily deliver results in real-time. A useful classification for online trackers can also be found in Smeulders et al. (2014), which also provides examples of algorithms that fit into each category.

The first category includes the most straight forward video trackers, the ones that simply try to match the object of interest with its pair in the next frame without recurring to any other techniques. Matching itself is the basis of all video tracking; it requires to establish a method to evaluate how similar two images are. The algorithms that are provided as representative of this category are: Normalized Cross-Correlation (Briechle & Hanebeck, 2001), Lucas-Kanade Tracker (Baker & Matthews, 2004), Kalman Appearance Tracker (Nguyen & Smeulders, 2004), Fragments-based Robust Tracking (Adam, Rivlin, & Shimshoni, 2006), Mean Shift Tracking (Comaniciu, Ramesh, & Meer, 2000), and Locally Orderless Tracking (Oron, Bar-Hillel, Levi, & Avidan, 2015).

The next category goes one step further, it contains trackers that use matching and also generate an extended appearance model. These algorithms try to learn from the object as time goes on and more frames are processed. This previous results help to create an extended model that helps to predict

certain behaviors of the object and improve the tracking results. This extended model is usually based on identifying certain features and how these behave in the overall composition of the object. Examples are: Incremental Visual Tracking (Ross, Lim, Lin, & Yang, 2008), Tracking on the Affine Group (Kwon, Lee, & Park, 2009) and Tracking by Sampling Trackers (Maggio & Cavallaro, 2011).

There are also trackers that are based on matching, but also try to use sparse optimization techniques to generate dynamic constraints that describe an sparse representation of the object. Sparse optimization is intended to provide approximations to the real solutions of certain function, this is achieved by finding the variables that have a greater effect on the result and gradually ignoring the ones of little impact. In this category one may find: Tracking by Monte Carlo sampling (Kwon & Lee, 2009), Adaptive Coupled-layer Tracking (Čehovin, Kristan, & Leonardis, 2011), ℓ_1-minimization Tracker (Mei & Ling, 2009) and ℓ_1 Tracker with Occlusion detection (Mei, Ling, Wu, Blasch, & Bai, 2011).

Another category contains the trackers that use discriminative classification by establishing the difference between tracked objects and background. Since the object has to be located previously, there is some implicit information regarding the background from the beginning. The background information can be used to search something that looks like the object of interest but also something that is different from the background. Some trackers that fit the description are: Foreground-Background Tracker (Nguyen & Smeulders, 2006), Hough-Based Tracking (Godec, Roth, & Bischof, 2013), Super Pixel tracking (Wang, Lu, Yang, & Yang, 2011), Multiple Instance learning

Tracking (Babenko, Yang, & Belongie, 2009), and "Tracking, Learning and Detection" (Kalal, Matas, & Mikolajczyk, 2010).

The last category is the one of trackers that mixes discriminative classification of background and constraints generated by sparse optimization, all done to improve the matching process. The only example provided is Struck, also known as Structured output tracking with kernels (Hare et al., 2016).

There are many video tracking algorithms that will not be mentioned, for further information on other video trackers and how each one works it is recommended to read: Trucco and Plakas (2006); Maggio and Cavallaro (2011); Smeulders et al. (2014); and Galoogahi et al. (2017). The only techniques that will be described in detail in following sections are the Sobel filter and the Zero Mean Normalized Cross-Correlation, these two were selected from the different tools for video tracking for the following reasons:

- These techniques are very simple in comparison with other techniques (Smeulders et al., 2014; Sobel & Feldman, 1968). Zero Mean Normalized Cross-Correlation is very similar to Normalized Cross-Correlation, this means that both belong to the simplest category of video tracking algorithms: the ones that only perform matching. The Sobel filter, on the other hand, can be considered a method to process the image before any actual operation.

- As it will be shown in section 3.5, Zero Mean Normalized Cross-Correlation is competitive against other video trackers in precision of the results (Smeulders et al., 2014). There are still better algorithms but those are increasingly complex in comparison.

- Since the objective of this thesis is to use Graphics Processing Units (section 2.1.1), the selected video tracker should be easy to parallelize. Zero Mean Normalized Cross-Correlation has the advantage of being based on summations and this type of operations are considered ideal for parallelization (Catanzaro, 2010).

- The selected video tracker is used along with the Honeybee Search Algorithm (section 1.2.3.1), this makes it a priority to select a video tracker that can be treated as an optimization problem (section 1.2.1). Zero Mean Normalized Cross-Correlation has the property of being mostly defined as an optimization problem (Bätz et al., 2014).

- Online video trackers are more interesting for this thesis. Zero Mean Normalized Cross-Correlation (just as Normalized Cross-Correlation) falls in this category.

1.1.1.1 Zero Mean Normalized Cross-Correlation

Zero Mean Normalized Cross-Correlation (ZNCC) is the name of a specific variation of Normalized Cross-Correlation (NCC), there is little difference between them. In fact, sometimes their names are even interchanged by researchers. As expected, ZNCC belongs to the first category that was mentioned in section 1.1.1, the simplest kind of algorithms that only try to match the object of interest with its pair. Even though it is one of the simplest approaches, it still has merit and can be compared to other more resent proposals in accuracy of results (Smeulders et al., 2014).

ZNCC is used to measure the correlation between two image templates.

In other words, ZNCC quantifies how similar these image templates are (Bätz et al., 2014; Di Stefano et al., 2005; Lewis, 1995). The grade of similarity is obtained as a scalar number, the greater it is, the more correlated these images are. Usually, the aim of using ZNCC is to find the point (u, v) where certain template t is located within certain image frame I (illustrated in Figure 1.2). In order to find the optimal (u, v), every suitable point (u, v) in I must be checked using ZNCC. The range of possible values for (u, v) is limited by the dimensions $w \times h$ of I as well as the dimensions $m \times n$ of t. The range of u extends from 0 to $w - m$ while the range of v goes from 0 to $h - n$.

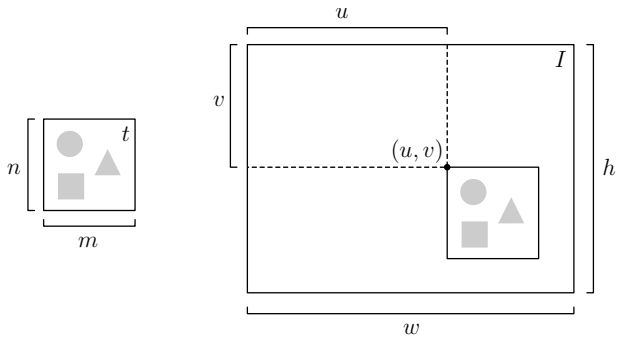

Figure 1.2: Template t can be contained in frame I in a rectangle of size $m \times n$ which has point (u, v) as top left corner.

The formula used to compute ZNCC for any given (u, v) is based on the squared Euclidean distance. Equation 1.1 shows how the squared Euclidean distance $d^2_{I,t}(u, v)$ may be obtained. The term $I(x, y)$ refers to the value used to describe the gray level of pixel located at point (x, y) within frame I. Similarly, $t(x-u, y-v)$ is the value of gray assigned to the pixel $(x-u, y-v)$ of template t. Note (x, y) is any point contained by the rectangular window

that has point (u, v) as the top left corner and $(u+m, v+n)$ as bottom right corner, meaning $u \leq x \leq u+m$ and $v \leq y \leq v+n$. The aforementioned inequalities ensure the top left corner of template t will always be $(0, 0)$, and, as expected, the bottom right corner will always be (m, n).

$$d^2_{I,t}(u, v) = \sum_{x,y} [I(x, y) - t(x-u, y-v)]^2 \quad (1.1)$$

$$d^2_{I,t}(u, v) = \sum_{x,y} I^2(x, y) - 2I(x, y)t(x-u, y-v) + t^2(x-u, y-v) \quad (1.2)$$

The expression used to obtain $d^2_{I,t}(u, v)$ may be expanded, as in equation 1.2. This manipulation reveals the result depends on 3 terms. The term $\sum_{x,y} t^2(x-u, y-v)$, which remains constant, is ignored since it does not change for two different points (u, v). On the other hand, the term $\sum_{x,y} I^2(x, y)$ does change for different values (u, v), but this change reflects the fact that the selected point is different rather than measuring the similarity between pixels $I(x, y)$ and $t(x-u, y-v)$. Only the term $\sum_{x,y} I(x, y)t(x-u, y-v)$ is useful to get the cross-correlation $c(u, v)$ (equation 1.3) and thus it is established as a starting point for ZNCC.

$$c(u, v) = \sum_{x,y} I(x, y)t(x-u, y-v) \quad (1.3)$$

ZNCC presents an improvement over $c(u, v)$, since the first provides resistance to changes across the frames of a video sequence. For example, a difference in the illumination of the targeted object that occurs between frames of the video. Another common problem that occurs when using $c(u, v)$

is that bright spots in the image or template can cause confusion by provoking greater correlation values than actual features of the targeted object. In order to give ZNCC a greater resistance to the mentioned errors, $c(u, v)$ is altered by normalizing the image and feature vectors to the unit length (this is reflected in equation 1.4). The mean of all the pixels (x, y) in the region were $u \leq x \leq u + m$ and $v \leq y \leq v + n$ is denoted as $\bar{I}_{u,v}$ and found using equation 1.5. The term $\bar{t}$ is the mean of the pixels contained in t and requires equation 1.6 to be evaluated.

$$\gamma(u, v) = \frac{\sum_{x,y} [I(x, y) - \bar{I}_{u,v}][t(x - u, y - v) - \bar{t}]}{\sqrt{\sum_{x,y} [I(x, y) - \bar{I}_{u,v}]^2 \sum_{x,y} [t(x - u, y - v) - \bar{t}]^2}} \tag{1.4}$$

$$\bar{I}_{u,v} = \frac{\sum_{x,y} I(x, y)}{m \times n} \tag{1.5}$$

$$\bar{t} = \frac{\sum_{x,y} t(x - u, y - v)}{m \times n} \tag{1.6}$$

The value $\gamma(u, v)$, called ZNCC, ranges between -1 and 1 (Bätz et al., 2014). As the value approaches 1, the certainty of finding t in point (u, v) with great robustness is greater. However, ZNCC is not capable of detecting certain transformations such as rotation and scale changes in the targeted object.

1.1.1.2 Sobel Filter

The Sobel filter or Sobel operator is used to detect the edges that are present on images, helping find points of special interest that characterize the picture in question. Edges can be described as boundaries that separate different

textures in the same image. These edges are found when there is an abrupt change from one pixel to another (Juneja & Sandhu, 2009; Patnaik & Yang, 2012; Shrivakshan, Chandrasekar, et al., 2012; Sobel & Feldman, 1968). When the Sobel filter is applied on an image, a second image is generated as output (see Figure 1.3). This second picture has brighter values on pixels that show greater contrast with their vicinity. These brighter pixels define features that help distinguish objects from one another. This filter was not initially proposed for video tracking, but can be used to emphasize the important characteristics of objects of interest.

Figure 1.3: Sobel filter applied on an image of a marble. Original image obtained from the ALOV++ dataset (Smeulders et al., 2014)

The Sobel filter is different from other edge detectors because it is based on the numerical analysis of 3×3 sized pixel neighborhoods and is used to detect changes in both horizontal and vertical directions (Juneja & Sandhu, 2009; Patnaik & Yang, 2012; Sobel & Feldman, 1968). Certain vicinity is centered on certain pixel (x, y) within an image I. The central pixel is surrounded by 8 neighbors, as shown in Figure 1.4. In order to apply the Sobel filter to pixel (x, y), a 3×3 matrix is composed using the values from the neighborhood and then convoluted with two different special matrices called kernels.

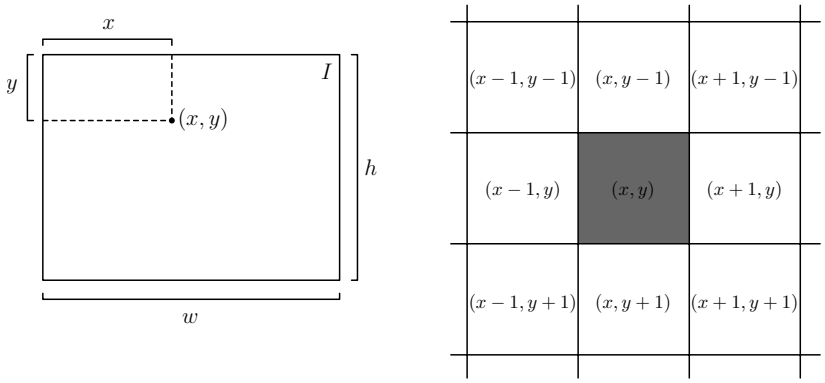

Figure 1.4: Pixel (x, y) has 8 neighbors.

Convolution consists in performing a special kind of matrix multiplication expressed as $h = I * g$, where h is the result of applying the kernel g of size $k \times k$ on the 2D signal I. In order to find the corresponding $h(x, y)$ for any given $I(x, y)$, equation 1.7 should be evaluated which results in obtaining a sum of the neighbors of (x, y) with certain ponderation effect caused by the weights defined by kernel g.

$$h(x, y) = \sum_{i=-k/2}^{k/2} \sum_{j=-k/2}^{k/2} I(x + i, y + j)g(i, j) \tag{1.7}$$

As mentioned earlier, in the special case of the Sobel filter $k = 3$ and the kernels are already defined and called S_x (equation 1.8) and S_y (equation 1.9). Since there are two different kernels, there are also two convolution operations that result in the gradients $\nabla_x = I * S_x$ and $\nabla_y = I * S_y$, that are computed using equations 1.10 and 1.11 respectively.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1.8}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1.9}$$

$$\nabla_x(x,y) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} I(x+i,y+j)S_x(i,j) \tag{1.10}$$

$$\nabla_y(x,y) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} I(x+i,y+j)S_y(i,j) \tag{1.11}$$

Since the kernels are relatively small (3×3), it is common to see the fully expanded sum forms to obtain $\nabla_x(x,y)$ and $\nabla_y(x,y)$, as shown in equation 1.12 and equation 1.13.

$$\begin{aligned} \nabla_x(x,y) = {} & I(x+1,y-1) + 2I(x+1,y) + I(x+1,y+1) \\ & -[I(x-1,y-1) + 2I(x-1,y) + I(x-1,y+1)] \end{aligned} \tag{1.12}$$

$$\begin{aligned} \nabla_y(x,y) = {} & I(x-1,y-1) + 2I(x,y-1) + I(x+1,y-1) \\ & -[I(x-1,y+1) + 2I(x,y+1) + I(x+1,y+1)] \end{aligned} \tag{1.13}$$

The absolute value of each gradient gives an estimate of the derivative of $I(x,y)$, but only on a horizontal (∇_x) or vertical (∇_y) direction. Considering this, both gradients can be combined to obtain the overall gradient

measurement applying equation 1.14.

$$\nabla(x,y) = \sqrt{\nabla_x^2(x,y) + \nabla_y^2(x,y)} \tag{1.14}$$

1.2 Algorithms Inspired in Honeybees

The honeybee (Apis mellifera) is one of the social insects of most interest for Swarm Intelligence (SI) researchers; this derives from the high level of organization that is observed in a honeybee swarm (Bitam et al., 2010; Karaboga & Akay, 2009).

Similar social conducts can be observed mainly on other insect species of the Hymenoptera order such as ants and wasps (also interesting for SI). There are complex altruistic relations between these individuals that produce what can be described as intelligent collective behavior, almost as if they were a unique superorganism (Nowak, Tarnita, & Wilson, 2010; Wilson, 1975). The described behavior is commonly known to biologists as eusociality, one of the most complex social structures in the animal kingdom (Crespi & Yanega, 1995).

The level of organization that was described is present in many of the activities of eusocial beings, and many of these activities can be seen as general optimization problems that computer science needs to solve. The clever ways in which these optimization problems are solved naturally serve as inspiration for many algorithms that wish to solve the same generalized situations.

The following section will provide an introduction to Swarm Intelligence and Evolutionary Computing, a description of the behavior of honeybees, an overview of the algorithms that have taken inspiration from them, and then a detailed description of the Honeybee Search Algorithm and why it was selected for this thesis.

1.2.1 Population Based Optimization

Both Swarm Intelligence (SI) and Evolutionary Algorithms (EAs) belong to a broader category: population based meta-heuristics (Bitam et al., 2010). One thing in common that all these different algorithms share is that they are all based on optimization problems.

The common optimization problem depends on certain function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, called the objective function (Luenberger, Ye, et al., 1984). An optimization problem also defines constraints as functions $g : \mathbb{R}^n \rightarrow \mathbb{R}^p$ and $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Any value in the domain of f, g or h is called x. Function g provides inequality constraints, since only the values of x that satisfy $g(x) \leq 0$ are acceptable. Similarly, h provides equality constraints as x should satisfy $h(x) = 0$. A value for x is called feasible if it meets the constraints. The set $\Omega = \{x \in \mathbb{R}^n : g(x) \leq 0, h(x) = 0\}$ that contains all feasible solutions is called the feasible region.

The general optimization problem, as defined earlier, may be summarized in one small statement as follows:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned}$$

The answer to this optimization problem should be some value x^* such that x^* is feasible and $f(x^*) \leq f(x)$ holds for any x. This value x^* is called an optimal value. The definition of an optimization problem does not restrict the objective function in many ways. Meaning f does not have to be differentiable or have any other property that helps to find the optimal value x^*. As a last resort, a common approach to find the optimal value is to compute all the values in the feasible region one at a time. The described approach can generate heavy temporal computational costs, and this is the reason why population based meta-heuristics have gained popularity. These meta-heuristics present alternative and intelligent methods to search for x^* without having to cover the whole feasible region.

1.2.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are based on concepts taken from biological evolution. In this context, a specific value for $x \in \Omega$ is called an individual and the objective function f is called the fitness function. A population is a group of individuals and a generation is the state of this population on a specific iteration of the EA. In each iteration, the population is evaluated (using the fitness function) and modified with the help of the genetic operators: mutation, crossover, and selection. The best individuals survive and generate offspring until a stop condition is met. This procedure explores the feasible

region with special emphasis on the best found solutions (Deb, 2001; Goldberg, 1989).

The purpose of the selection operator is to give the best individuals a greater chance to generate offspring for the next generation. This means selection is mainly concerned with evaluating the individuals using the fitness function. One of the most common and simple selection operators is the tournament selection. In tournament selection, two individuals are selected randomly from the population. This two individuals are compared using the fitness function and the best one gets to be part of the list for crossover (or the next generation in some cases). The number of tournament matches depends on how many individuals need to be selected; this depends on the type of EA.

Mutation and crossover operators depend on the type of EA. There are many different kinds of EAs, but the following paragraphs will only describe Genetic Algorithms and Evolution Strategies because of their importance in order to describe how the Honeybee Search Algorithm works.

Genetic Algorithms Genetic Algorithms (GAs) differ from other EAs mainly in their binary representation of individuals (Deb, 2001; Goldberg, 1989). Each individual is represented by an array of bits and each bit is called a gene. Binary representation is attractive because it simplifies mutation and cross-over operators. Mutation can be accomplished by randomly selecting a gene and negating its value. Binary crossover usually receives two individuals as input (called parents) and outputs two new individuals (called children). The most common binary crossover is one-point crossover where both parents

are split at a random point. The pieces of both parents are then reassembled to create the children as illustrated in Figure 1.5.

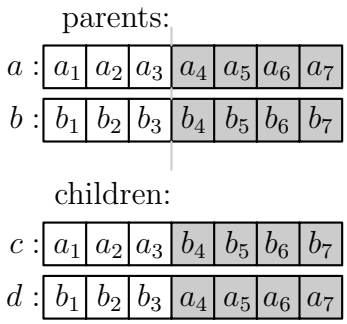

Figure 1.5: Binary crossover of parents a and b, generating children c and d.

Evolution Strategies Evolution Strategies (ESs) are another type of EAs. In this case, individuals are represented as n-dimensional vectors composed of real valued components to resemble the actual solutions of the optimization problem. ESs do not commonly use crossover, mutation is the biggest source of change between generations.

The mutation operator can be easily applied by adding normally distributed random values to the individual x (see equation 1.15). The normal distribution N used by mutation has median 0 and standard deviation σ. In this context, σ is called mutation strength, since increasing this value allows greater mutations to the individuals.

$$y_i = x_i + N(0, \sigma) \tag{1.15}$$

There are other mutation operators that can be used with ESs. For

example, Polynomial Mutation can be achieved using equations 1.16 and 1.17. There has to be a random number $u \in [0, 1)$ and values x_i^U and x_i^L have to be defined using the upper and lower boundaries of the individual's i-th component x_i (Khare, Yao, & Deb, 2003). The mutated individual y is expected to be close to x, but how often depends on the constant parameter η_m.

$$\bar{\delta}_i = \begin{cases} (2u_i)^{\frac{1}{\eta_m+1}} - 1 & \text{if } u_i < 0.5 \\ 1 - [2(1-u_i)]^{\frac{1}{\eta_m+1}} & \text{otherwise} \end{cases} \tag{1.16}$$

$$y_i = x_i + (x_i^U - x_i^L)\bar{\delta}_i \tag{1.17}$$

As mentioned earlier, ESs do not commonly use crossover but it also is possible to implement. For example, the Simulated Binary Crossover (SBX) operator is designed to have similar properties to the crossover performed in Genetic Algorithms but using real valued variables $x \in \mathbb{R}$ (Deb, 2001; Deb & Beyer, 1999; Khare et al., 2003). A random value β_q is used to obtain children c_1 and c_2 from parents x_1 and x_2 using equations 1.18 and 1.19.

$$c_1 = 0.5[(1 + \beta_q)x_1 + (1 - \beta_q)x_2] \tag{1.18}$$

$$c_2 = 0.5[(1 - \beta_q)x_1 + (1 + \beta_q)x_2] \tag{1.19}$$

In order to select an specific β_q using equation 1.20, a random number u that ranges from 0 to 1 is selected. This same procedure can be applied by each component of the parent individuals. There is also a constant η_c that

helps to modify the frequency of having children that are near to parents.

$$\beta_q = \begin{cases} (2u)^{\frac{1}{\eta_c+1}} & \text{if } u \leq 0.5 \\ \left[\frac{1}{2(1-u)}\right]^{\frac{1}{\eta_c+1}} & \text{otherwise} \end{cases} \tag{1.20}$$

Another thing that distinguishes ESs is that the population has a constant size of μ individuals across generations. As each generation passes, a new population of λ individuals is generated as offspring. The next generation can be selected from both the μ and λ populations or only from the λ population. The constitution of the next generation depends on the kind of ES, some examples of ES types are:

- $(\mu + \lambda)$-ES: The next generation is conformed by the best individuals from both the μ and λ populations, this means parents compete with the offspring (this is called elitism).
- (μ, λ)-ES: The next generation is conformed by the best individuals from the λ population exclusively.

1.2.2 The Natural Behavior of Honeybees

Since honeybees are called eusocial beings, they share very specific characteristics with other eusocial species. According to Crespi and Yanega (1995), said characteristics are:

- for an animal species to be eusocial, adult individuals of the species have to cooperate to take care of the younger individuals in the community;

- there has to be a division of work based on reproduction, meaning there is a royal or reproductive caste and a worker caste that is partially or totally sterile;
- several generations have to live together in the same colony and contribute to fulfill their common needs.

As illustrated in Figure 1.6, the royal caste of honeybees is composed by all the male bees and the only female capable of laying eggs, the queen bee (Von Frisch, 2014). In contrast, the worker caste is conformed by all the remaining female bees. The differences between female bees of different castes are not only social but also physical. Since their early life, queen bees get special treatment. They are fed royal jelly which helps them increase their body size and fully develop their ovaries thanks to the presence of a special protein called royalactin (Kamakura, 2011). This quicker and greater development distinguishes them from other female bees, which suffer from malnourishment during their larval stage.

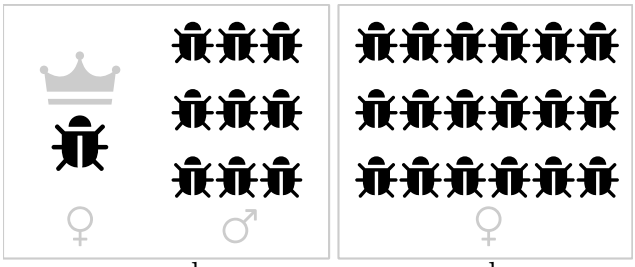

Figure 1.6: Honeybees are divided in castes: the royal or reproductive caste and the worker caste. All the males and the queen honeybee belong to the royal caste and all the other female honeybees belong to the worker caste.

The queen bee has other functions in the hive besides reproduction. For

instance, it is in charge of coordinating the activities of the worker bees through the regulated production of special pheromones that cause changes in the conduct and labors of all the bees that live in the hive (Slessor, Winston, & Le Conte, 2005). The queen bee is also in charge of keeping the balance in the population, since it is capable of deciding the sex of offspring. This is possible because, in honeybees, the unfertilized eggs become male bees through parthenogenesis[1] while the fertilized ones always become female bees (Tucker, 1957; Von Frisch, 1955).

The other members of the royal caste, the males, are also called the drones because of their clumsy and lazy nature. Their main and only task in the hive is to mate with queen, in order to increase the population of worker bees. It is common that drones mate with the queen from other hives and even are welcome to join those other hives (Von Frisch, 1955). Most of their distinguishable traits are developed primarily for the purpose of mating, which occurs during flight and outside of the hive. They have larger eyes and more olfactory sensors on their antennae to be able to track down the flying queen (Schlüns, Koeniger, Koeniger, & Moritz, 2004). The mating process is always fatal to them, and if they happen to survive long enough, they are expelled from the hive during autumn when their task has been fulfilled.

Worker bees earn their name because they are in charge of most of the maintenance tasks in the hive such as nursing larvae, guarding the hive entrance, honey production, foraging, etc. Their main labor depends on their age and development. They begin their adult stage by cleaning and taking care of larvae. As they begin to learn how to fly and develop certain

[1]Parthenogenesis: a kind of asexual reproduction, it happens when the female produces eggs that develop without ever being fertilized (Tucker, 1957).

special organs their tasks become more specialized, from honey and royal jelly production to protecting the entrance to their home with their stings. The oldest worker bees are ones who leave the hive in search for food and resources, this work is so harsh and dangerous that all of them will die (at most) some weeks later (Von Frisch, 1955).

The specific task of searching for and collecting nectar, pollen and other resources is performed by worker bees that take two different roles: explorers and foragers. Explorers leave the hive when they perceive there is need to search for new food sources and return when they have located one. Forager bees are then recruited by explorers to harvest certain food sources. The information brought by explorers is used to make decisions regarding resource allocation. The better a food source is, the more foragers will be recruited to exploit it (see Figure 1.7).

Figure 1.7: Explorer bees communicate their findings using their dance language. The best food sources get proportionately more recruits.

Explorers can tell foragers about the location, quality and quantity of the food they have found with great detail through their dance language. There

is a distinction between two types of dances: the round dance and the waggle dance (Crist, 2004). The round dance is used when the resources are close enough and only the direction needs to be communicated. On the other hand, the waggle dance is used for longer distances, and the speed of the dance is related to the distance that has to be traveled, a faster dance means a closer distance. The direction that must be taken during the flight can be acquired from the angle of the dancing bee, which relates to the angle between the sun and the food source. The possible recruits can also confirm the quality of the source by smelling samples brought by the explorer. And finally, the quantity of food is related to the liveliness of the dance.

1.2.3 Overview of Honeybee-Inspired Algorithms

As explained earlier in this chapter, the organization that honeybees show in their daily activities has inspired many researchers to propose different algorithms for several problems that originate from computer science. This section provides small introductions to some notable proposals that have this common source of inspiration. Bitam et al. (2010) initially proposed to categorize several different algorithms based on what specific activity of the honeybee hive is emulated. The categories that were proposed are:

1. algorithms inspired in the search for food sources of honeybees
2. algorithms inspired in the search for a new nest site of honeybees
3. algorithms inspired in the marriage behavior of honeybees

The first two models are similar, since honeybees perform the same search process (described in section 1.2.2) in both cases, where the communication

and coordination trough their dance language is vital. The main difference between this two models is that when honeybees search for food sources more than one sources need to be found to sustain the hive; when honeybees wish to find a new nesting site, there has to be an overall consensus, the discussion goes on until only one option is left because the efforts to build a new hive cannot be started until then. This means that the first model is used when the optimization problem of interest can or must have several different answers, while the second approach is used when the problem requires only a unique optimal solution.

The marriage behavior model has certain similarities with Evolutionary Algorithms, because it is based on how the individuals that show a greater fitness for their environment are the ones that get to produce offspring. As mentioned in section 1.2.2, the queen honeybee is the only female capable of reproduction and has many potential mates. These mates have to show their fitness by competing with all other male honeybees in the hive, characteristics that help them to overcome their test are greater vision and flying speed to be able to track down and reach the queen during mating flights.

The first category is the most populated, distinguished examples are:

- Bee System (Lucic & Teodorovic, 2001) which was initially tested with several instances of the Traveling Salesman Problem (combinatorial optimization), and several benchmarking problems.

- Bee Colony Optimization (Teodorovic & Dell'Orco, 2005) is an improvement of the original Bee System. The main difference is that this algorithm was capable of dealing with uncertainty in combinatorial optimization problems, by using fuzzy logic.

- Honey Bee Algorithm (Nakrani & Tovey, 2003) emulates how honeybees self-organize in order to dynamically allocate internet services in a highly unpredictable environment (like the real Internet).

- BeeHive (Wedde, Farooq, & Zhang, 2004) also applies the model to the problems of networking. In this case it is applied to perform routing in wired networks.

- BeeAdhoc (Wedde et al., 2005) could be seen as another version of BeeHive, but focused on energy efficient routing for mobile ad hoc networks.

- Artificial Bee Colony (Karaboga, 2005) was initially proposed for multi-dimensional and multi-modal optimization problems, but can be adapted to be used for combinatorial optimization.

- Virtual Bee Algorithm (Yang, 2005) can be used to solve numerical optimization problems. It is specialized in continuous functions and was tested with De Jong's test function (single-peaked) and the Keanes multipeaked bumpy function.

- Bee Algorithm (Pham et al., 2011) is also used for combinatorial and numerical optimization. It was used to train Learning Vector Quantization networks to recognize patterns in control charts with problems that had 2,160 parameters.

- The Honeybee Search Algorithm (Olague & Puente, 2006a, 2006b) which was used to solve the problem of three-dimensional reconstruction

from a stereo pair of images. It will be described in detail in section 1.2.3.1 because it is used in the experiments.

- OptBees (Maia, de Castro, & Caminhas, 2012) is characterized by maintaining the diversity of population to obtain and evaluate many local optima. This algorithm was tested with 5 of 20 problems proposed by the Optimization Competition of Real Parameters of the CEC 2005 Special Session on Real-Parameter Optimization.

- Mutable Smart Bee Algorithm (Mozaffari, Gorji-Bandpy, & Gorji, 2012) uses "Mutable Smart Bees" instead of common honeybees. These individuals are capable of remembering the history of the visited locations and quality of food sources; they also have a small chance of mutation as individuals in Evolutionary Algorithms.

The second category has a lower number of examples; food search based algorithms are much more popular in the literature. An example of the second model can be found in Quijano and Passino (2010), their main interest was to solve resource allocation problems as a numerical optimization. They were able propose ideal free distribution and globally optimal allocation strategies. The third model also has a lesser number of examples. One specific example is the Bees Mating Optimization algorithm (Abbass, 2001) that was initially used to solve the famous NP-complete problem of three-satisfiability (3-SAT).

The next section will describe the specific algorithm that is used in this work, the Honeybee Search Algorithm. This algorithm was selected because it has already been successfully used for Computer Vision, as mentioned earlier (Olague & Puente, 2006a). Other interesting qualities of the algorithm

are that: it combines concepts from Evolutionary Algorithms and Swarm Intelligence; it belongs to the most popular category of honeybee-inspired algorithms, the ones based on the search for food; and it is not used specifically for combinatorial optimization problems, giving a greater freedom to define the fitness function and search space. Another important factor that influenced the decision was the availability of an older implementation of the Honeybee Search Algorithm, used for reference.

1.2.3.1 The Honeybee Search Algorithm

The Honeybee Search Algorithm was initially proposed by Olague and Puente (2006a, 2006b), it can be defined as a meta-heuristic that combines Evolutionary Algorithms and Swarm Intelligence with individuals that emulate the behavior of foraging honeybees. It is inspired in the search process performed by explorer and forager honeybees that was described in section 1.2.2. There are three main phases: exploration, recruiting and harvest, as displayed in Figure 1.8. The exploration phase consists in sending search agents called explorers to random points of the fitness function, these points or possible solutions are called food sources. The next phase, recruiting, is where the explorers return to the hive and try to convince other bees to follow them and exploit the food source they have found. In the last phase, harvest, a massive search is executed where the surroundings of the best food sources are given a greater attention since more bees are attracted by them.

The exploration stage (Figure 1.9) can be described with greater detail as a modified ES of the type $\mu + \lambda$ (more information about $(\mu + \lambda)$-ESs available in section 1.2.1.1). This kind of Evolutionary Algorithm is commonly

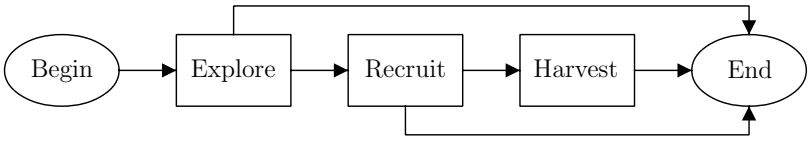

Figure 1.8: The Honeybee Search Algorithm has 3 phases: exploration, recruitment and harvest.

distinguished by the importance of mutation as the main source of change between generations (instead of crossover). But, in this case, the λ_e sons are generated by 3 different methods: α_e sons are generated by Polynomial Mutation, β_e sons are generated by crossover using SBX and γ_e sons are generated randomly. It should be noted that $\alpha_e + \beta_e + \gamma_e = \lambda_e$.

```
Exploration(μ_e, λ_e)
 1  Initialize(μ_e)
 2  Evaluate(μ_e)
 3  while stop condition = false
 4      Generate(λ_e)
 5      Evaluate(λ_e)
 6      Sharing(μ_e, λ_e)
 7      Select μ_e best(μ_e, λ_e)
```

Figure 1.9: Pseudocode for the exploration phase.

Another important difference between the common ES and the one used in this algorithm is that a sharing operator is also used to penalize individuals that concentrate in a small space by reducing the individual's fitness value. This thesis does not use the sharing operator, so no further details are provided. For a formal explanation of this operator, the lecture of Olague and Puente

(2006a) is recommended.

The recruiting phase is simpler; the total number λ of forager agents is divided into groups. Just as illustrated in Figure 1.7, the size r_i of those groups is relative to the fitness value of each of the individuals that were selected from the exploration phase, as seen in equation 1.22. In order to find r_i, a proportion p_i is computed using equation 1.21, dividing the corresponding fitness value fit_i by the sum of all fitness values obtained in the exploration phase (μ_e in total).

$$p_i = \frac{fit_i}{\sum_{j=1}^{\mu_e} fit_j} \tag{1.21}$$

$$r_i = p_i \times \lambda \tag{1.22}$$

The harvest phase (Figure 1.10) is also a $(\mu + \lambda)$-ES where the offspring is generated in the exact same way as in the exploration phase ($\lambda_h = \alpha_h + \beta_h + \gamma_h$). The differences are that, instead of beginning with random points, the starting points are the results from the exploration phase; and the number of generations and individuals may change. This means the algorithm performs an ES search for every good food source found during exploration.

```
Harvest(μ_h, λ_h, μ_e)
 1   for each individual ∈ μ_e
 2       Initialize(μ_h)
 3       Evaluate(μ_h)
 4       while stop condition = false
 5           Generate(λ_h)
 6           Evaluate(λ_h)
 7           Sharing(μ_h, λ_h)
 8           Select μ_h best(μ_h, λ_h)
```

Figure 1.10: Pseudocode for the harvest phase.

Chapter 2

Framework about the Parallelization of Swarm Intelligence Algorithms

The following chapter will provide an introduction to the concept of Parallel Computing, along with relevant concepts such as Graphics Processing Units (GPUs). This will be done to provide a framework, or a set of tools, that can be used to parallelize Swarm Intelligence (SI) algorithms such as the Honeybee Search Algorithm.

Section 2.1 will be mainly focused on the description of parallel computing technologies. GPUs will get more attention than other parallel computing technologies, as the experiments to be described in Chapter 4 are performed using this specific type of technology.

The next section, section 2.2, is where the general difficulties of implementing SI algorithms in GPUs will be discussed, along with several approaches that have been used in the literature with similar purposes to the ones of this

thesis.

2.1 Parallel Computing

The definition of Parallel Computing is the usage of some set of processors that work in collaboration to solve a single computational problem (Foster, 1995). It is an opposite notion to sequential computing. Traditionally, a computational problem is solved by breaking it down into smaller trivial problems such as addition, multiplication, logical operations, etc., these are executed in order one at a time until an answer is obtained. In parallel computing, a problem is also broken down, but some instructions can be executed simultaneously by different processors and the results are later combined by a coordination mechanism (Figure 2.1). The reason why Parallel Computing has gained popularity in recent years is the ever increasing size of computational problems that require solution, while time to deliver results is always required to decrease.

Different technologies have emerged from this trend such as Graphics Processing Units (GPUs), clusters, grids, Field Programmable Gate Arrays (FPGAs), etc. clusters and grids are similar, they are both based on the idea of using common computational resources interconnected by networking technologies (Hwang, Dongarra, & Fox, 2013). GPUs take a different approach; their computational resources are embedded in a single hardware for very specific applications. FPGAs loosely fit the definition of Parallel Computing technologies because it is possible to implement highly complex logic circuits with them, these might as well take the form of parallel computing resources

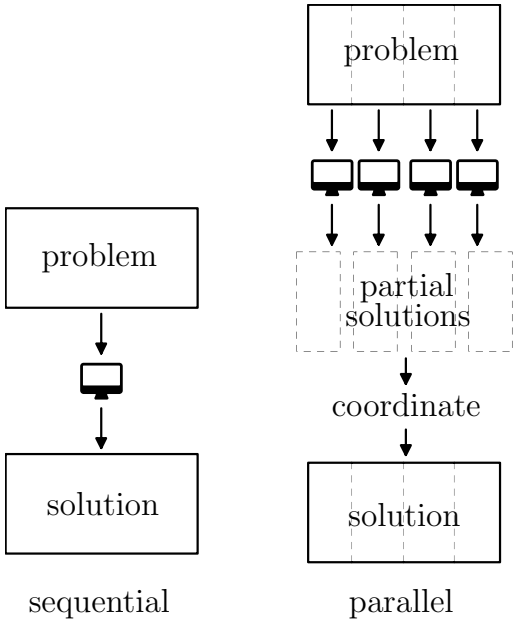

Figure 2.1: Parallel computing divides the problem in smaller problems that are solved by many computational resources; partial solutions are generated and later combined.

(Draper, Beveridge, Bohm, Ross, & Chawathe, 2003). Since there are so many different kinds of parallel computers, a simple classification system was proposed by Flynn (1972). which is commonly known as Flynn's taxonomy. This taxonomy has 4 different classifications for parallel computers which are based on how they perform instructions over data:

- Single-instruction stream – single-data stream (SISD): This category is used to describe common computers, only one instruction is performed once over certain data.

- Single-instruction stream – multiple-data stream (SIMD): Includes parallel computers that perform several operations at a time over different

input data, but all the operations have to execute the same instruction. This is especially useful for graphic processing applications where matrix operations are required. Depending on the architecture, GPUs can be included in this category or at least their main components.

- Multiple-instruction stream – single-data stream (MISD): This is a very specialized approach that can be found mainly on hardware specifically designed for signal processing. Different operations are performed in parallel over the same input data.

- Multiple-Instruction stream – multiple-data stream (MIMD): Clusters fall in this category. Each processor is capable of performing different instructions on different input data, achieving full parallelism for general purpose. If FPGAs are used for parallel computing, they commonly fit here (Dhaussy, Filloque, Pottier, & Rubini, 1994; Raimbault, Lavenier, Rubini, & Pottier, 1993).

2.1.1 Graphics Processing Units

A Graphics Processing Unit (GPU) can be seen as an array of hundreds of processing units that are designed for quick performance on graphics rendering, mainly three-dimensional graphics (Owens et al., 2008). However, as soon as tools for programming were made available by vendors, the scientific community started to use GPUs for general purpose and research. There are several Application Programming Interfaces (APIs) such as CUDA (Compute Unified Device Architecture), which is only for NVIDIA products (Tan & Ding, 2016); and the Open Computing Language abbreviated as OpenCL

(Owaida, Bellas, Daloukas, & Antonopoulos, 2011). These APIs allow the implementation of programs for GPUs, but the decision of which to use is most influenced by the manufacturer of the specific product. The interest in using this technology for research has had an effect on the evolution of the GPU; today's products are built with different architectures that allow general purpose applications (Mantor, 2012; Owens et al., 2008).

As mentioned earlier, GPUs can either be considered SIMD devices or be made of SIMD components, depending on the specific model. The instructions can be performed by different SIMD components, sometimes called vector units. How data is handled in GPUs is another characteristic, commonly there is a global memory that can be read or written by any processing units (Owens et al., 2008). There is also local memory that is only available to certain processing units; this is added because using global memory can become a bottle neck when many different processing units require access to the same data at the same time.

The main coordination mechanisms that are observed in GPUs are performed by the common CPU. Since the main purpose of GPUs is to help in image rendering, these are connected to a common CPU so that normal operations of the computer are performed by it. The GPU is not usually required to have an internal synchronization or coordination mechanism because the CPU is the 'leader' that takes these responsibilities. This can be problematic if the program that is implemented in a GPU has data dependencies between different phases (Tan & Ding, 2016).

```
∑_{x_i∈X} x_i(X, N)
1   Σ = 0
2   for i = 0 to N − 1
3       Σ = Σ + x_i
```

Figure 2.2: Pseudocode for sequential summation. Σ retains the result of the summation.

2.1.2 Parallel Reduction of Summations

As mentioned by Nickolls, Buck, Garland, and Skadron (2008), there are certain problems based on having a sequence of N numeric values that have to be combined by some operation. These problems are called reduction problems, and have properties that allow them to be easily parallelized. A classic example is the summation of the N elements of certain vector X; if this problem is solved sequentially the program would be similar to the pseudocode of Figure 2.2, where x_i is i-th element of vector X. It does not matter how fast the processor that uses the sequential procedure is, it will always take around N steps to get a result.

The specific problem of summation is nicely behaved, because addition has the properties of associativity and commutativity (Catanzaro, 2010; Chandra, 2001). Said properties allow the distribution of the operations in an inherently parallel manner. If two computational resources were available, the problem could be treated as two different summation problems $a = \sum_{i=0}^{N/2-1} x_i$ and $b = \sum_{i=N/2}^{N-1} x_i$. Both of those summations give a partial answer to the full problem of obtaining the sum $c = \sum_{i=0}^{N-1} x_i$ and are called partial sums. The reduction process is quite simple $a + b = c$ but the time reduction is evident,

while the original sequential algorithm takes N steps, the parallel reduction with two computational resources should take $N/2$ steps.

Using only two computational resources could not be enough in most cases, but the idea can be generalized to use any number p of different computational resources and solve the problem in almost N/p steps. The general idea of reduction is having many different pieces of data and reducing them into a single result, but parallel reduction does this by dividing the load of work, as illustrated in Figure 2.3. Each computational resource performs the reduction operations in its own section of the set, then each of the individual results are reduced into one single result; in the case of summation, the total of adding all the numbers in X.

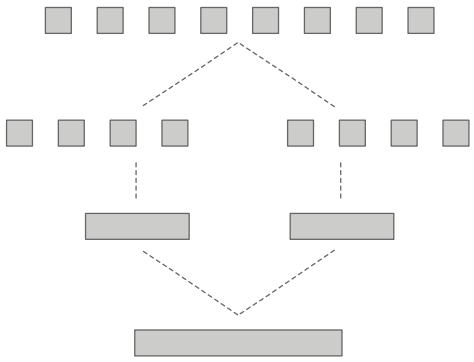

Figure 2.3: Parallel reduction consists in performing operations on many pieces of data by dividing them in groups, once those groups of data are reduced to single pieces of data; these are reduced into a unique result.

Generally speaking, parallel reduction can be used whenever certain problem requires a large number of operations, but not any kind of operation. As clarified by Solihin (2015), some operations that can take advantage of parallel reduction are sum, product, maximum, minimum, and logical

operations (and, or, xor). A good example of a problem that can use this parallel reduction is ZNCC; note equation 1.4 is mostly based on summations. Section 3.2 will use this paradigm to parallellize the computation of said function.

2.2 Parallelization of Swarm Intelligence Algorithms with Graphics Processing Units

Since one of the objectives of this thesis is to parallelize a Swarm Intelligence (SI) algorithm, the Honey bee Search Algorithm, it is useful to analyze how some of these algorithms have been parallelized in the past and described in the literature. It is even more useful to analyze how common SI algorithms have been parallelized using GPUs, the selected parallel computing technology to be used by this thesis. Tan and Ding (2016) have proposed a list of models that have been identified by observing several parallel implementations of SI algorithms specifically with the technology of interest. According to the mentioned work, when a SI algorithm is parallelized using a Graphics Processing Unit (GPU), said implementation has to fall in one of the following 4 categories:

1. naive parallel model
2. multiphase parallel model
3. all-GPU parallel model
4. multiswarm parallel model

This list can be seen as a progression, rather than mutually exclusive categories. Multiphase parallel model does what the naive parallel model does but goes one step further, the same is true for the next categories. Each step tries to solve the problems that the previous model faces until the best possible parallelization approach is achieved.

The naive parallel model is the simplest approach when trying to parallelize a SI; it treats the fitness functions as black boxes. The common SI simply needs to provide individuals as input to the black box and it only has to deliver the corresponding fitness values. The process performed by the black box is not really of importance, as long as fitness values are correctly computed. If the problem can be seen this way, the fitness function itself can be easily parallelized, since it is already an optimization problem with a given function $f(x)$. Each process that is executed in parallel can be in charge of computing $f(x)$ for certain x. An attractive characteristic of this model is that it is completely independent from the type of SI, and even can be used when several SIs want to be tested with the same fitness function. Examples are: the Particle Swarm Optimization (PSO) implementation from Kalivarapu and Winer (2008); PSO implementation by Cardenas-Montes, Vega-Rodriguez, Rodriguez-Vazquez, and Gomez-Iglesias (2011); the PSO-based tool of Hsieh and Chu (2011); and the Cellular Automata (CA) urban model through cooperative coevolutionary Particle Swarms proposed by Blecic, Cecchini, and Trunfio (2014).

When the naive model is not enough to improve the time costs of certain SI algorithm, further parallelization can be implemented with the multiphase parallel model. It consists in parallelizing other tasks that are commonly

required such as: sorting, finding minimal or maximal values, performing large sums, roulette-wheel selection, etc. Even though many activities of the SI are performed in the GPU, the CPU still is in charge of keeping track of the current state of the SI algorithm and sending the GPU the data to perform these very specific tasks. This could be a problem, because communication between the CPU and GPU is relatively slow and can be considered a bottle neck that should be avoided. There is another factor that affects the time cost when using this model, each time the GPU is instructed to execute certain program the CPU has to send the full source code and the GPU has to compile it. If many different routines are performed, this will inefficiently perform the compilation process every time the routine changes. Some implementations that use this model are: the Ant Colony Optimization (ACO) with tabu search from Tsutsui and Fujimoto (2011); the Bees Algorithm implementation from Luo, Huang, Chang, and Yuan (2014); and the PSO implementation from Zhou and Tan (2009).

All-GPU parallel model is the next step to take when the parallelization of the multiphase model is not enough. Every single SI operation is done in the GPU, in a way that only initial data is transferred from the CPU to the GPU and the final results are delivered to the CPU, which becomes only a bridge between the user and the GPU. When there is code that cannot be parallelized it is preferable to simply perform it in one of the computational resources that are available rather than waiting for the CPU–GPU communication. There is a common problem with this approach: when synchronization is needed for certain activities it cannot be achieved because hardware is simply not designed to do so. Synchronization in GPUs is only possible between groups

of processes, that belong to the same block. This means only a fraction of the available resources can be used with this model. Mussi, Nashed, and Cagnoni (2011) proposed a PSO that fits this model; also Calazan, Nedjah, and de Macedo Mourelle (2013) describes a similar approach, also using PSO.

Finally, the multiswarm parallel model is an answer to the synchronization problem of the all-GPU parallel model, but it cannot be solved completely. There is a way to use all the resources of the GPU: each block of processes can behave as an independent SI instance. This means, multiple all-GPUs happen simultaneously and once all of them are done, the CPU is in charge of searching in the results for the actual optimal results. There are many examples, some are: Swarm grid (Calazan, Nedjah, & de Macedo Mourelle, 2012) which is based on PSO; The PSO implementation proposed by Zhao, Wang, Pedrycz, and Tian (2012); and Cooperative Evolutionary Multi-Swarm Optimization Algorithm (Souza, Teixeira, Monteiro, & de Oliveira, 2012).

As can be noticed in the following chapters, the approach that was taken for the parallelization of the Honeybee Search Algorithm is the all-GPU parallel model. This is done as a first step to implement the multiswarm parallel model in future work.

Chapter 3

Video Tracking Using the Parallel Honeybee Search Algorithm

The following chapter describes the methodology used to track a targeted object across the various frames of the same video sequence by using the Parallel Honeybee Search Algorithm. The Parallel Honeybee Search Algorithm, as other Swarm Intelligence (SI) algorithms, can change its application with relative ease by presenting certain problem as an optimization problem which depends on certain objective function. In the specific case of video tracking, there are many candidate functions that can be treated as an optimization problem for this purpose. The selected objective function for this research is a combination of the Sobel filter and the Zero mean Normalized Cross-Correlation (ZNCC), both popular tools in the field of computer vision. Commonly, as mentioned by Tan and Ding (2016), the evaluation of the objective function is one of the most time consuming tasks for SI algorithms

and some parallelization efforts are centered specifically on the adaptation of the objective function for parallel computing.

With that in mind, the methodology is divided in five stages. The first two stages (sections 3.1 and 3.2) will help to understand how the Sobel filter and ZNCC were combined and justify the approach taken for parallelization. The Parallel version of ZNCC combined with the Sobel filter will also serve as a control group that will be useful to detect any effect caused by the usage of the Parallel Honeybee Search Algorithm over the brute force (or extensive search) approach. The third and fourth stages (sections 3.3 and 3.4) provide an explanation of how the Honeybee Search Algorithm was adapted for video tracking and provide the details of the Parallel Honeybee Search Algorithm implementation to allow any future reproduction. The fifth stage (section 3.5) will describe the F-score and how it helps to evaluate the accuracy of the results given by any video tracking algorithm. This metric will be useful to show if there is an important difference in accuracy caused by the usage of the Parallel Honeybee Search Algorithm.

3.1 Analysis of the Sobel Filter and Zero Mean Normalized Cross-Correlation for Video Tracking

The Sobel filter and Zero mean Normalized Cross-Correlation (ZNCC) are both common tools in the field of computer vision. The two of them are also related by their common inspiration on concepts which are used widely

for signal processing and pattern recognition in several fields such as cross-correlation and convolution (Kapinchev, Bradu, Barnes, & Podoleanu, 2015; Ratha, Jain, & Rover, 1995). This section provides further details on how both are used in combination for this research's purposes.

The overall idea is to replace the original image frame I and template t with new images generated by applying the Sobel filter on I and t respectively. The new pictures would emphasize important detectable features which are potentially useful for video tracking. ZNCC may take advantage of the information provided by the Sobel filter by being used to find the level of correlation between the filtered template and the filtered image frame.

Both, ZNCC and the Sobel filter, are designed to be used with greyscale images, but occasionally useful information about the features can be obtained from color. A common way to adapt these tools to be used with digital color images is to repeat the process for each color channel and combine them following certain criteria. The majority of digital pictures have 3 color channels: red, green and blue. In the case of the Sobel filter, this research chooses to simply combine the 3 separate results by sum (equation 3.1). The total gradient $G_I(u, v)$ helps detect edges on picture I at pixel (u, v), by combining the gradients $\nabla_{Ir}(u, v)$, $\nabla_{Ig}(u, v)$ and $\nabla_{Ib}(u, v)$, each being obtained as indicated by equation 1.14 using the corresponding color channel of image I.

$$G_I(u, v) = \nabla_{Ir}(u, v) + \nabla_{Ig}(u, v) + \nabla_{Ib}(u, v) \tag{3.1}$$

Having established how to obtain $G_I(u, v)$, the same procedure could be used to get $G_t(u, v)$ by replacing the input image with template t. By

modifying equation 1.4 to find the correlation between the gradients rather than I and t one obtains equation 3.2. Also note that the averages $\overline{G}_I$ and $\overline{G}_t$ need to be found previously using equations 3.3 and 3.4.

$$\gamma_G(u,v) = \frac{\sum_{x,y}[G_I(x,y) - \overline{G}_I][G_t(x-u,y-v) - \overline{G}_t]}{\sqrt{\sum_{x,y}[G_I(x,y) - \overline{G}_I]^2 \sum_{x,y}[G_t(x-u,y-v) - \overline{G}_t]^2}} \tag{3.2}$$

$$\overline{G}_I = \frac{\sum_{x,y} G_I(x,y)}{m \times n} \tag{3.3}$$

$$\overline{G}_t = \frac{\sum_{x,y} G_t(x-u,y-v)}{m \times n} \tag{3.4}$$

3.2 Parallel Implementation of the Sobel Filter and Zero Mean Normalized Cross-Correlation

The following section serves as a description of the approach used to paralellize the computation of the fitness function used for video tracking in this research. Said fitness function is the combination of ZNCC and Sobel filter $\gamma_G(u,v)$ as defined in equation 3.2. The description of the parallelization begins with an abstract theoretical explanation in section 3.2.1. It is followed by section 3.2.2, which provides details specific to the implementation used with a Graphics Processing Unit (GPU).

3.2.1 Parallelization of the Sobel Filter and Zero Mean Normalized Cross-Correlation

First, a distinction must be made between two different cases when the parallelization is attempted. The main difference of both cases is based on their prior knowledge of the total number of values for (u, v) to be checked and how its magnitude compares to the number of computational resources available.

As suggested in section 1.1.1.1, the number of possible values for (u, v) depends on the dimensions $w \times h$ of I as well as the dimensions $m \times n$ of t. The feasible region Ω can be defined as $\{(u, v) \in \mathbb{Z}^2 : 0 \leq u \leq w - m, 0 \leq v \leq h - n\}$ and has cardinality $|\Omega| = (w - m) \times (h - n)$.

If the number of available computational resources p is expected to be significantly lesser than $|\Omega|$, a reasonable approach for parallelization is to divide the workload by assigning each computational resource the evaluation of $\gamma_G(u, v)$ for certain section of Ω (see Figure 3.1). On the other hand, if p is expected to be significantly greater than $|\Omega|$, or just a limited subregion of Ω is of interest, then several computational resources should work in coordination to compute $\gamma_G(u, v)$ for an specific value of (u, v).

A useful generalization for both approaches is to consider how an specific $\gamma_G(u, v)$ is computed by q computational resources working in coordination. The first logical step is to evaluate $\overline{G}_I$ and $\overline{G}_t$ using equations 3.3 and 3.4. In said equations, sums are the most time consuming operations. For example, to evaluate the sum $\sum_{x,y} G_I(x, y)$ using only one computational resource, $m \times n$ terms are obtained and then added one by one. But having q computational

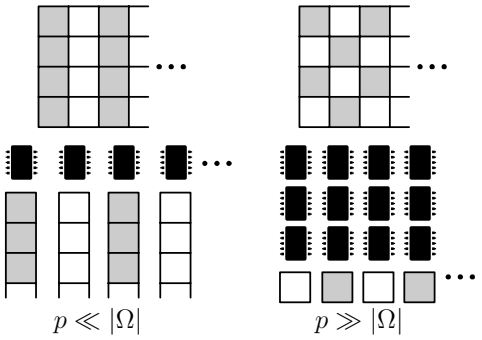

Figure 3.1: The division of work depends on $|\Omega|$ and p. If the number of tasks $|\Omega|$ is greater than the number p of computational resources, then each one is in charge of many tasks. If there are fewer tasks than computational resources, then each task is performed by several of these. Each square represents a task.

resources at disposal partial sums can be evaluated simultaneously and then reduced to a single result. Said partial sums would be the result of adding only $(m \times n)/q$ terms. This is illustrated in Figure 3.2, where each square represents a task and has an identifier x_i and each computational resource (labeled with identifier q_i) is in charge of $(n \times m)/q$ tasks.

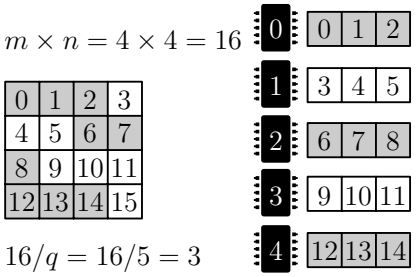

Figure 3.2: Each task (square) has an identifier x_i. Each computational resource is in charge of $(m \times n)/q$ tasks. Note the task with id 15 is not assigned; it should be the responsibility of the first or last computational resource.

If each of the q computational resources are assigned an integer identifier

q_i in the interval $[0, q-1]$ the partial sum of the q_i-th computational resource is obtained using equation 3.6. An integer x_i used to identify each pixel to be used by the q_i-th computational resource can be found using equation 3.5. Observe $u + x_i(k) \mod m$ is the corresponding x, while $v + x_i(k) \div m$ is the corresponding y of the given pixel (x, y) for which $G_I(x, y)$ is being computed and accumulated. Finally, once all partial sums have been evaluated, a single computational resource may obtain $\sum_{x,y} G_I(x, y)$ by adding q terms. Ideally, that single computational resource should be prepared to take any remaining workload that was not been taken care of.

$$x_i(k) = k + q_i \frac{mn}{q} \tag{3.5}$$

$$\sum_{k=0}^{(mn/q)-1} G_I \left[u + x_i(k) \mod m, v + x_i(k) \div m \right] \tag{3.6}$$

The described procedure may be repeated to parallelize the evaluation of the sum used to get $\overline{G}_t$ and the rest of sums required to compute $\gamma_G(u, v)$ with small modifications. The overall procedure goes as shown in the pseudocode of Figure 3.4; where instructions outside the if statement use all q computational resources as described, and instructions inside the if statement use only one of the computational resources while the rest wait. Said procedure would only obtain $\gamma_G(u, v)$ for certain (u, v). In order to evaluate several values for (u, v) simultaneously, p (the total number of computational resources) should be a multiple of q. That relation would allow the evaluation of $\gamma_G(u, v)$ for p/q different pixels at the same time by p/q different 'teams' of q computational resources each as shown in Figure 3.3.

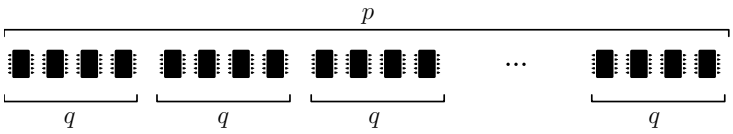

Figure 3.3: The p computational resources are divided in teams of size q to evaluate $\gamma_G(u, v)$ for p/q different pixels simultaneously.

Parallel $\gamma_G(u, v, q_i)$
1 Compute partial sum of $\sum_{x,y} G_I(x, y)$
2 Compute partial sum of $\sum_{x,y} G_t(x - u, y - v)$
3 **if** $q_i = 0$
4 Compute $\sum_{x,y} G_I(x, y)$
5 Compute $\sum_{x,y} G_t(x - u, y - v)$
6 Compute $\overline{G}_I$
7 Compute $\overline{G}_t$
8 **else**
9 Wait
10 Compute partial sum of $\sum_{x,y} [G_I(x, y) - \overline{G}_I][G_t(x - u, y - v) - \overline{G}_t]$
11 Compute partial sum of $\sum_{x,y} [G_I(x, y) - \overline{G}_I]^2$
12 Compute partial sum of $\sum_{x,y} [G_t(x - u, y - v) - \overline{G}_t]^2$
13 **if** $q_i = 0$
14 Compute $\sum_{x,y} [G_I(x, y) - \overline{G}_I][G_t(x - u, y - v) - \overline{G}_t]$
15 Compute $\sum_{x,y} [G_I(x, y) - \overline{G}_I]^2$
16 Compute $\sum_{x,y} [G_t(x - u, y - v) - \overline{G}_t]^2$
17 Compute $\gamma_G(u, v)$
18 **else**
19 Wait

Figure 3.4: Pseudocode to compute $\gamma_G(u, v)$ in parallel. This algorithm is executed by several GPU processes, each identified by the id q_i. The instructions inside if statements are executed only by certain GPU processes. The rest wait for that process to finish and then keep going.

3.2.2 Parallel Implementation of the Sobel Filter and Zero Mean Normalized Cross-Correlation with a Graphics Processing Unit

Two different parallelized programs that obtain the combination of Sobel filter and ZNCC have been implemented to be used in this research. The first one obtains $\gamma_G(u, v)$ for all the feasible region Ω, while the second one only finds $\gamma_G(u, v)$ for a given list of values for (u, v). Since the feasible region Ω is commonly of a greater size than the number of processes p that the GPU can execute, the first program assumes $q = 1$, so that only one process is used to compute $\gamma_G(u, v)$. In contrast, the second program uses $q > 1$ and requires coordination between processes. The reason the second program was developed is to serve as a component of the Parallel Honeybee Search Algorithm. Table 3.1 provides a summary of the characteristics of both implementations.

Table 3.1: Summary of the characteristics of each implementation

Version	**Size of teams**	**Type of Search**	**Coordination**
1 (NO BEE)	$q = 1$	Full feasible region	None
2 (BEE)	$q > 1$	Honeybee Search	Required

Both versions were implemented using the C++ programming language and OpenCL (Open Computing Language), an open source industry standard for parallel computing programming to be used with CPUs, GPUs, DSPs, etc. (Stone, Gohara, & Shi, 2010). The implementations also use OpenCV (Open Computer Vision) an open source computer vision library (Bradski &

Kaehler, 2008).

The usage of a GPU comes with certain advantages, as well as some difficulties. For instance, the shared memory architecture of the common GPU allows all processes to have access to the information of any pixel in the stored image data, simplifying the process of deciding which data should be sent and where it should be sent. On the other side, GPUs have several restraints regarding memory, mainly the restrictive size of global memory, in addition to the limited number and size of variables that can be declared. Said restraints had an impact on the implementation, since some of the evaluated video sequences had remarkably heavy file sizes.

To deal with the memory size problem, the image frames had to be reduced in size using the CPU before being sent to the GPU for further processing. This was implemented using OpenCV's function `cv::resize`, used to change the size of a picture to fit a new width and height. In order to ensure a certain level of detail the reduction of the size of the image is dependent on the size of the template to be searched rather than the size of the full image. As illustrated in Figure 3.5, the new size of the image depends on a previously established constant called `max_window`. If the width of the template (m) is greater than its height (n) then the image is reduced so that $m =$ `max_window`, in the opposite case the new size should be such that $n =$ `max_window`.

Another problem that arose from the decision of using a GPU along with OpenCL was related to the need of coordination in the second version of the program. The processes in a GPU are performed in a relatively asynchronous manner. When using OpenCL, it is not possible to synchronize all the processes (called work-items) that are available to the GPU because there

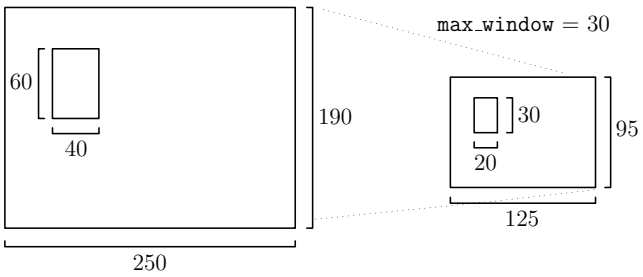

Figure 3.5: The images are reduced using the constant `max_window`. In this case, the height n is greater than the width so the new height is equal to `max_window`.

is no warranty these are working at the same time. It is only possible to synchronize the work-items that belong to the same work-group (Owaida et al., 2011). This means full synchronization is only possible when using a small number of work-items which is the maximum size of a work-group (this number depends on the specific GPU model). This type of synchronization uses a special instruction called barrier. When a barrier instruction is added to an OpenCL program all work-items have to stop and wait until every member of the work-group has executed the barrier instruction.

In summary, both versions of the parallel program need to reduce the size of the image before the GPU actually starts working. The problem of coordination that limits the number of processes that can be used is unique to the second version, since the first version does not require any kind of synchronization. These details could have potentially unfavorable effects on the results but, given the technology requirements, these solutions had to be implemented.

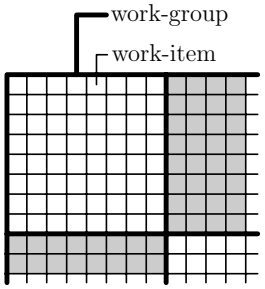

Figure 3.6: The work-items of a GPU are arranged in work-groups. Coordination is only possible for the work-items (processes) that belong to the same work-group.

3.3 Analysis of the Honeybee Search Algorithm for Video Tracking

The simplest and most evident change made to the Honeybee Search Algorithm so it can be used for video tracking is to use function $\gamma_G(u, v)$ as described in equation 3.2 to evaluate fitness of individuals. This means the respective optimization problem shall be defined as follows:

$$\begin{aligned} \text{Maximize} \quad & \gamma_G(u, v) \\ \text{subject to} \quad & 0 \leq u \leq w - m \\ & 0 \leq v \leq h - n \end{aligned}$$

Where each individual (u, v) must belong to the feasible region $\Omega = \{(u, v) \in \mathbb{R}^2 : 0 \leq u \leq w - m, 0 \leq v \leq h - n\}$. The individual (u, v) may also be interpreted as the pixel (u, v) that marks the beginning of the rectangular window suspicious of containing template t in image frame I, an illustration

can be found in Figure 1.2.

The exploration phase of the algorithm is left almost intact as the pseudocode shown in Figure 1.9, except for the sharing operator. The original Honeybee Search Algorithm was applied with a different goal, the three-dimensional reconstruction of a scene based on a pair of stereo images (Olague & Puente, 2006a). In said application the desired result was a cloud of three-dimensional points rather than a unique individual. As a consequence, the sharing operator was of great help to ensure the variety of solutions suggested by the algorithm giving a greater number of usable points to the resulting 3D model. But in the case of video tracking, the expected result is a unique individual (u, v) that indicates the most probable location of a given object. With this in mind, the sharing operator was excluded.

To continue, the recruitment phase was modified to limit the section of the feasible region to be searched in a different fashion. Originally, small search spaces were defined around each food source found during the exploration phase (as in the left part of Figure 3.7). Then, the recruited foragers would search on their corresponding search space which was defined by the final location of their specific recruiter. But this was changed so that a unique subsection of the feasible region is defined. This single sub-region (as shown in the right part of Figure 3.7) is defined so that all explorers can be contained within it. This does not affect the starting point for forager bees, the better a food source is the more forager bees are recruited by the explorer that found it. The number of forager bees that start in that location is still found using equation 1.22.

Finally, the harvest phase was also affected by the exclusion of the sharing

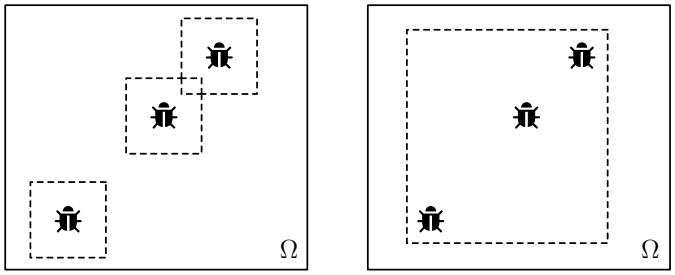

Figure 3.7: The search region is reduced during the recruitment phase. The left image shows how the original Honeybee Search Algorithm defined a smaller search region around each recruiter. The right image shows how the current implementation defines a unique region that contains the recruiters.

operator, just as the exploration phase was. Except for that, the harvest phase is almost identical to the pseudocode that can be found in Figure 1.10.

3.4 Parallel Implementation of the Honeybee Search Algorithm

The usage of a GPU and OpenCL brought some limitations to the implementation of the Honeybee Search Algorithm. Some of those problems and the respective solutions are discussed in section 3.2.2. But one last problem that had a negative effect on the implementation of the Honeybee Search Algorithm directly is discussed in this section.

While it is true that a GPU has many processors in comparison with a common CPU, it is also true that these processors have a reduced function set. This presents certain difficulties such as the lack of native support for random number generation. This could not be ignored, since the Honeybee

Search Algorithm requires random numbers for several vital activities such as: Polynomial Mutation, SBX, tournament selection and random search. The solution that was implemented was to pre-generate random number arrays of different distributions with the CPU and send them along with the frame image data to the GPU. Four arrays are generated: 1) uniformly distributed integers (using OpenCV's `cvRandInt`), 2) uniformly distributed real numbers (using OpenCV's `cvRandReal`), 3) beta distribution (equation 1.20) and 4) delta distribution (equation 1.16).

Moving on to the description of the actual implementation, several changes had to be done in order achieve parallelization. The parallel evaluation of the fitness function works as described in section 3.2 of this same chapter. Since a number q of processors is used to find a specific value of $\gamma_G(u, v)$, from now on this will be seen as each individual bee being composed of q GPU processors (Figure 3.8).

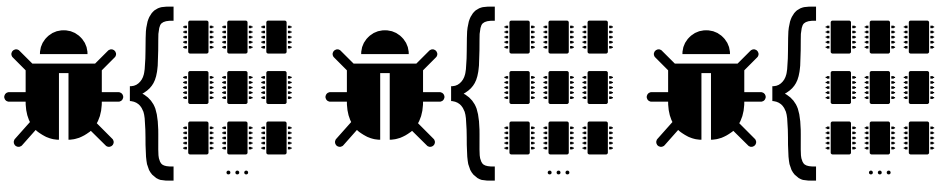

Figure 3.8: Each individual is made of a number of GPU processors.

The maximum number of individuals that can work simultaneously is p/q. To take advantage of these p/q individuals, all must be used at once, there is no benefit in using any less and using more would cause a dramatic increase in time cost. With this in mind, the size of populations μ_e, λ_e, μ_h and λ_h were all set to the same size so that $\mu_e = \lambda_e = \mu_h = \lambda_h = p/q$. This is a

notable difference with the original sequential algorithm, where commonly $\mu_e < \lambda_e$ and $\mu_e < \mu_h$ with the intention of correctly emulating the behavior of honeybees, where the resources for initial exploration are significantly lesser than the ones for harvest.

When evaluating the fitness function, all processors that belong to certain bee have work to do. But this is not the case all the time, there are activities in the Honeybee Search Algorithm that require a lower level of parallelization and only use one processor per bee (Figure 3.9). The list of activities that fall in this category are:

- generation of the initial random μ_e population for the exploration phase
- generation of λ populations using mutation, crossover and random exploration
- merge μ and λ populations
- selection of the μ best from $\mu + \lambda$ populations

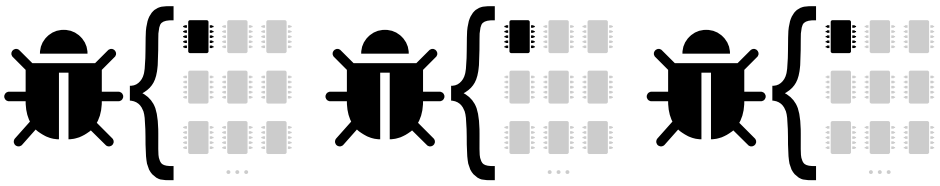

Figure 3.9: Some activities only require one processor per bee.

When the λ population is generated, each bee is in charge of the generation of one specific son. The type of son generated by each bee depends on its global id. The first α bees generate their respective offspring using Polynomial

Mutation. The next β bees are generated by crossover using SBX and γ sons are generated randomly, but respecting the search space constraints.

Before the selection of the μ best from the $\mu+\lambda$ population, said population has to be sorted by fitness value. The sorting algorithm that was used in the described implementation is a parallelized merge-sort, that uses 1 processor per bee. The described sorting algorithm is very similar to the one used in Davidson, Tarjan, Garland, and Owens (2012), except that the one used by them has a flexible number of elements for each sublist. An example of how this parallel merge-sort works can be seen in Figure 3.10. Since each bee behaves both as μ and λ bees, the representative processor of each bee is in charge of sorting its local list, but this is a trivial case since said list has only two elements. Once local lists are sorted, two lists are merged by one processor. Only half of the active processors are required to merge lists so half of the processors are sent to rest. As the sorting process continues the number of active processors is divided by 2, while the merged lists grow twice in size. This goes on until there are only two lists to merge and only a single processor is required sort them.

The only phase that was not parallelized at all is the recruitment phase. One processor keeps working while the rest remain idle. This processor alone assigns how many foragers will be recruited by each explorer bee and define the new search space as described in the last section (3.3). The harvest phase is very similar to the exploration phase except for three main changes: The number of generations in the harvest phase is only half of the number used in the exploration phase because the search space is already reduced significantly; while the initial population for the exploration phase is random, the initial

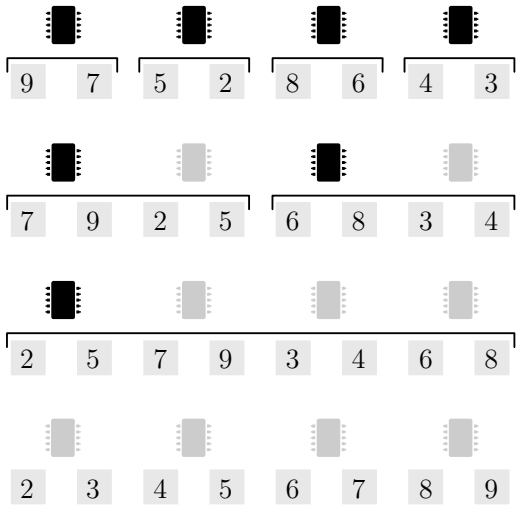

Figure 3.10: The fitness values are sorted using merge-sort, initially each processor is in charge of sorting two elements. Lists grow twice in size while half of the processors are set to rest in each step.

population for the harvest phase is located on the food locations found during the exploration phase with greater density on the best ones; and the search space is only a fraction of Ω, as established during the recruitment phase. The overall algorithm follows the pseudocode shown in Figure 3.11, where q_i is the identifier of the processor and b_i is the identifier of the individual.

```
Parallel Honeybee Search Algorithm(q_i, b_i)
 1  if q_i = 0
 2      Initialize(μ_e)
 3  else
 4      Wait
 5  while stop condition = false
 6      Evaluate(μ_e)
 7      if q_i = 0
 8          Generate(λ_e)
 9      else
10          Wait
11      Evaluate(λ_e)
12      if q_i = 0
13          Merge(μ_e, λ_e)
14          Sort(μ_e, λ_e)
15          Select μ_e best(μ_e, λ_e)
16      else
17          Wait
18  if q_i = 0 ∧ b_i = 0
19      Assign recruits(μ_e, μ_h)
20      Reduce search space(μ_e)
21  else
22      Wait
23  while stop condition = false
24      Evaluate(μ_h)
25      if q_i = 0
26          Generate(λ_h)
27      else
28          Wait
29      Evaluate(λ_h)
30      if q_i = 0
31          Merge(μ_h, λ_h)
32          Sort(μ_h, λ_h)
33          Select μ_h best(μ_h, λ_h)
34      else
35          Wait
```

Figure 3.11: Pseudocode of the Parallel Honeybee Search Algorithm. The exploration phase goes from line 1 to 17. Recruitment from 18 to 22. Harvest from 23 to 35. The if statements are used to limit the number of GPU processors that work on certain task.

3.5 Description of the F-Score as an Evaluation Metric for Video Tracking Algorithms

The F-score metric is of interest for this research since it enables to measure how accurate a certain video tracking algorithm is and helps to compare it against others. Figure 3.12 shows how this metric has been used to compare various trackers that were previously described in in section 1.1.1, these 19 algorithms are labeled as shown in Table 3.2 to avoid writing the full names every time they are mentioned. These different algorithms were tested using the 314 videos provided by the Amsterdam Library of Ordinary Videos also known as ALOV++ (Smeulders et al., 2014). This type of graph is called a survival curve, it shows the results sorted in descending order and is more common on the field of medicine to compare how different treatments behave on patients (Collett, 2015; Lawless, 2011). The curves that are closer to the horizontal line that is labeled as 1 can be considered the most accurate.

The test videos that were used to evaluate all the algorithms shown in Figure 3.12 are freely available and were used by this research to evaluate the parallel implementation of the function $\gamma_G(u, v)$ and the Parallel Honeybee Search Algorithm against each other, and against the algorithms already evaluated. To do that, the F-score F must be evaluated using equation 3.7 and 3.8. The first equation (3.7) is known as the PASCAL criterion and helps to identify false positives and true positives. The criterion depends on how the rectangular window T^i that is given as result for certain frame (also called

Table 3.2: Labels for video tracking algorithms

Algorithm	**Label**
Normalized Cross-Correlation	NCC
Lucas-Kanade Tracker	LKT
Kalman Appearance Tracker	KAT
Fragments-based Robust Tracking	FRT
Mean Shift Tracking	MST
Locally Orderless Tracking	LOT
Incremental Visual Tracking	IVT
Tracking on the Affine Group	TAG
Tracking by Sampling Trackers	TST
Tracking by Monte Carlo	TMC
Adaptive Coupled-layer Tracking	ACT
ℓ_1-minimization Tracker	L1T
ℓ_1 Tracker with Occlusion detection	L1O
Foreground-Background Tracker	FBT
Hough-Based Tracking	HBT
Super Pixel tracking	SPT
Multiple Instance learning Tracking	MIT
Tracking, Learning and Detection	TLD
Struck: Structured output tracking with kernels	STR

truth) intersects with the ground truth GT^i and how this area compares to the union of both as illustrated in Figure 3.13. The ground truth data was also provided as part of the ALOV++ dataset to allow faster testing and measurement of new algorithms.

$$\frac{|T^i \cap GT^i|}{|T^i \cup GT^i|} \geq 0.5 \tag{3.7}$$

The number of false positives n_{fp} increases when the PASCAL criterion is not met; the number of true positives n_{tp} increases on the opposite case; and the number of false negatives n_{fn} increases when the object is present on

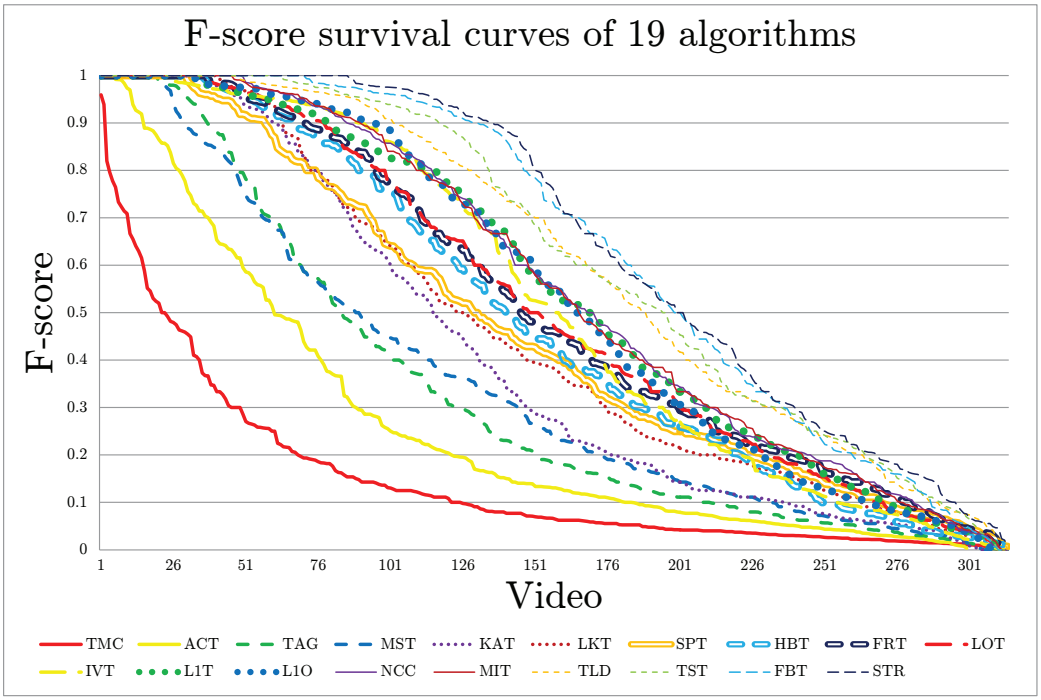

Figure 3.12: The F-score survival curve of 19 video tracking algorithms. This information was obtained from the Amsterdam Library of Ordinary Videos (ALOV++) provided by Smeulders et al. (2014).

the frame according to the ground truth, but the tested algorithm determines the object was not detected.

$$\begin{aligned} precision &= \frac{n_{tp}}{n_{tp} + n_{fp}} \\ recall &= \frac{n_{tp}}{n_{tp} + n_{fn}} \\ F &= 2 \cdot \frac{precision \cdot recall}{precision + recall} \end{aligned} \tag{3.8}$$

Note F only has non negative values lesser or equal to 1 and the PASCAL

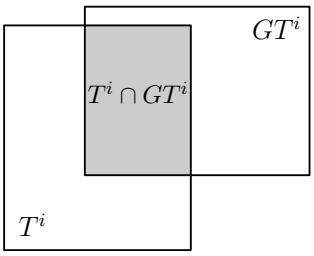

Figure 3.13: T^i (truth) and GT^i (ground truth) may intersect. The PASCAL criterion helps to quantify how similar T^i and GT^i are, based on this.

criterion measures not only that the result is in the correct place but also that it reflects the current correct size of the tracked object. The change in size is not considered at all by ZNCC, so this a weakness of the algorithm. Still, observe how NCC was able to keep a relatively good score in Figure 3.12 even though it is (also) not concerned with detecting the scaling of the object.

Chapter 4

Experiments Using a Graphics Processing Unit

The present chapter deals with further information about the implementation and testing of The Parallel Honeybee Search Algorithm applied for video tracking using the methodology that was discussed with detail in Chapter 3. It also provides specifics about the specific hardware that was used in this research; mainly the Graphics Processing Unit (GPU) that was used to implement the Parallel Honeybee Search Algorithm; and the CPU model used for preprocessing and displaying data. Other concerns of this chapter include the description of the folder and file structures of the Amsterdam Library of Ordinary Videos for tracking (ALOV++ dataset); and the configuration parameters that were used to perform the experiments with the Parallel Honeybee Search Algorithm implementation. The results of the described experiments are also presented.

For this purpose, the current chapter's content is divided in four parts. The first part, section 4.1 will document the important features of the hard-

ware that was used to perform the experiments, a brief explanation of the architecture of the GPU is also given. To continue, section 4.2 gives the introduction to the peculiarities of the ALOV++ dataset and how it was used to test and evaluate the accuracy and time cost of the Parallel Honeybee Search Algorithm for video tracking. Finally, sections 4.3 and 4.4 cover the experiments themselves, how these were performed and what results were yielded by them, in that order.

4.1 Description of the Hardware

A summary of the properties of the hardware that is described can be found in Table 4.1. The Graphics Processing Unit (GPU) that was used in the experiments that are about to be described is an AMD Radeon R9 270. It can be used for development through DirectX, Mantle, OpenGL, and OpenCL. The clock speed of this GPU is of 925 MHz and the communication with the CPU is made through a PCI Express x16 slot that transmits about 16 GB/s.

The AMD Radeon R9 270 is based on the Graphics Core Next (GCN) Architecture. As illustrated in Figure 4.1, said architecture divides the GPU in a number of Computing Units (CUs). These CUs are the building blocks of GPUs and are composed of four vector units and a single scheduler (Mantor, 2012). Every vector unit is an arrangement of 16 Arithmetic Logic Units (ALUs). This means, the CU has 64 ALUs in total. The AMD Radeon R9 270 has 20 CUs, which give a total of 1,280 ALUs.

The number of ALUs that are contained in the CU is not the same as the maximum size of the work-group (described in section 3.2.2), but

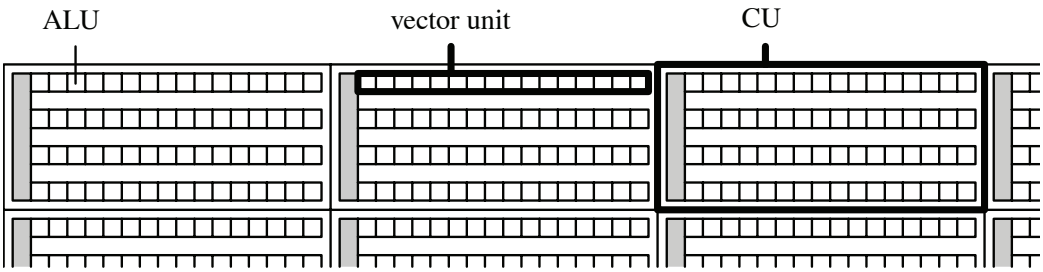

Figure 4.1: The processors of the GPU are arranged in Compute Units (CUs). Each CU has 4 vector units and each vector unit has 16 ALUs. There are 20 CUs in the AMD Radeon R9 270 (1,280 ALUs).

both concepts are related. In this specific GPU, the query of the property `CL_DEVICE_MAX_WORK_GROUP_SIZE` indicates the maximum size of the work-group is of 256 work-items. As noted by Gaster, Howes, Kaeli, Mistry, and Schaa (2012), the CU would need at least 192 work-items (3 per ALU) to use the full capabilities of the hardware, but the maximum of 256 cannot be exceeded in any GPU with the GCN architecture.

The version of OpenCL that was installed in the AMD Radeon R9 270 is known as OpenCL 1.1 Mesa 12.0.6, which allows the usage of double precision floating point variables with the inclusion of the `cl_khr_fp64` extension. The global memory can be read and written from any CU and has a size of 1 GiB, this is according to the `CL_DEVICE_GLOBAL_MEM_SIZE` property query. The full size of global memory is usable, but the maximum size of a single variable is of 0.25 GiB (according to the `CL_DEVICE_MAX_MEM_ALLOC_SIZE` property).

The CPU that was used to provide instructions to the GPU, image pre-processing (explained in section 3.2.2), random number generation (explained in section 3.4) and displaying results is a DELL XPS 8700 with an Intel

Core i7-4790. The base frequency of the processor is of 3.60 GHz. As mentioned in section 3.2.2, OpenCV (Open Computer Vision) is used for image manipulation, version 3.2 was installed in the CPU.

Table 4.1: Summary of the characteristics of the hardware

GPU

Model	AMD Radeon R9 270
Clock speed	925 MHz
ALUs	1,280
Max work-group	256 work-items
Global memory	1GiB

CPU

Model	Intel Core i7-4790
Clock speed	3.60 GHz

4.2 The Amsterdam Library of Ordinary Videos

The Amsterdam Library of Ordinary Videos, also called ALOV++ dataset, was initially proposed by Smeulders et al. (2014) to be able to test several video tracking algorithms with an experimental approach. The video sequences and ground truth data of the ALOV++ dataset are available to be used for research, as an example, these were used by Held, Thrun, and Savarese (2016) to train deep regression networks for video tracking. The efforts of the creators of the dataset resulted in a useful comparison of the accuracy of 19 video tracking algorithms (see Figure 3.12). Their main focus was the accuracy of different online trackers, other important information such as

the time taken to process each frame were not reported. This was achieved by gathering a considerable number of video sequences that present certain circumstances that obstruct the ability of many tracking algorithms to deliver trustworthy results. In total, 314 videos were gathered and organized in 14 categories. Each category presents an specific challenging situation for video tracking algorithms. These categories are:

1. **Light**: 33 videos that have sudden and intense changes on the main source of light or how the tracked object is illuminated by it.

2. **Surface Cover**: 15 videos where the tracked object changes its surface cover, but this cover adopts the form of the object it covers.

3. **Specularity**: 18 videos with shiny objects that reflect light and produce specularities.

4. **Transparency**: 20 videos where the tracked object is transparent and easily confused with the background.

5. **Shape**: 24 videos in which the tracked object changes its shape in a drastic manner.

6. **Motion Smoothness**: 22 videos that show an object that moves so slowly that no movement is detected at all.

7. **Motion Coherence**: 12 videos where the tracked object does not follow a predictable route of movement.

8. **Clutter**: 15 videos with tracked objects that display similar patterns to the ones observed in the background.

9. **Confusion**: 37 videos that show objects that are very similar to the object of interest, causing confusion.

10. **Low Contrast**: 23 videos in which the tracked object shows little contrast with the background or other objects.

11. **Occlusion**: 34 videos where the object of interest is occluded by other objects or is not in the field of vision at all at certain point.

12. **Moving Camera**: 22 videos that are affected by the sudden movements of the camera.

13. **Zooming Camera**: 29 videos where the zoom of the camera changes the displayed size of the object.

14. **Long Duration**: 10 videos that have greater durations, between one and two minutes.

The files are organized in folders named as their respective categories. Each frame of the video sequence is provided as an individual image file. The ground truth data is also provided in a separate folder that is also organized by category. These files contain plain text that can be easily interpreted. See Figure 4.2 for an example; note each line has an integer number followed by 8 floating point numbers. The first number indicates the frame, and the following numbers are 4 points $a = (a_x, a_y)$, $b = (b_x, b_y)$, $c = (c_x, c_y)$ and $d = (d_x, d_y)$. These points are the vertexes of a rectangle that encloses the object of interest.

In the case of the first line that is show in Figure 4.2, $a = (226.67, 58.571)$, $b = (139.52, 58.571)$, $c = (139.52, 148.57)$ and $d = (226.67, 148.57)$. See Figure

```
1 226.67 58.571 139.52 58.571 139.52 148.57 226.67 148.57
6 257.64 46.803 170.49 46.803 170.49 136.8 257.64 136.8
11 206.85 74.674 120.32 74.674 120.32 163.43 206.85 163.43
```

Figure 4.2: Example of the ground truth file structure of the ALOV++ dataset. Only the first three lines of the file `01-Light_video00023.ann` are shown.

4.3 to visualize how this points limit the bounding box that contains the object of interest. This information can be easily translated to the notation used by the fitness function $\gamma_G(u, v)$ (from equation 3.2). The template t that is going to be searched in the next frame has size $m = a_x - b_x$ and $n = c_y - b_y$ and begins at the point $(u, v) = (b_x, b_y)$. The point (u, v) is also useful as an indicator of where the Parallel Honeybee Search Algorithm should start the quest for the best match in the next frame.

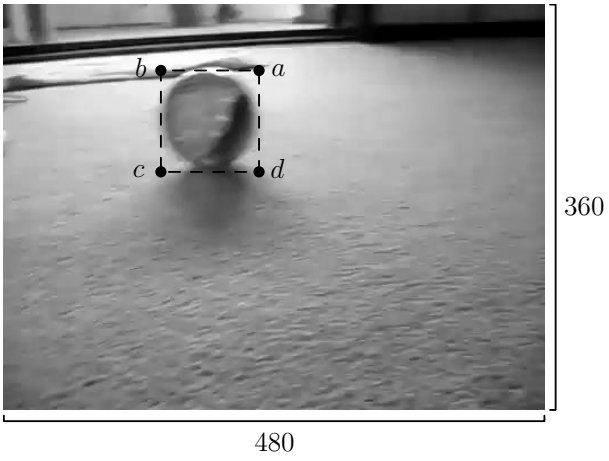

Figure 4.3: Example of the ground truth bounding box. This image is the first frame of the 23th video sequence of the Light category. Points a, b, c and d are obtained reading the file displayed in Figure 4.2.

The experiments done by this research use the information taken from the first line of every ground truth file to initialize the algorithm, so the first frame is not taken into account when the F-score is calculated. Note that the example file of Figure 4.2 takes a long step from frame 1 to frame 6, and later to frame 11. The size of this step also changes between video sequences, but there is no video that has ground truth information for every frame. In the case where the object is not visible, the file simply does not provide any information of the given frame. Remembering the concepts of false positive (n_{fp}), true positive (n_{tp}) and false negative (n_{fn}) that were introduced in section 3.5; these can only be detected on certain frame when there is information about the ground truth on said frame. The false positive (n_{fp}) detection could be problematic if it is supposed to be detected when the video tracking algorithm 'finds' the object of interest in a frame where it is not actually visible according the ground truth, this was not considered since the step from one frame to another in the ground truth data is not always constant.

The implementation of the Parallel Honeybee Search Algorithm used in the experiments looks for the object in every frame, even if there is no ground truth data of said image frame. It's also important to note that the algorithm has no mechanism to detect the absence of the object of interest in the current frame; this means that a false negative (n_{fn}) is not possible.

4.3 Configuration Parameters used in the Experiments

Two different programs where developed to be used in the experiments. This was first mentioned in section 3.2.2 and Table 3.1 summarizes the characteristics of each program. The first program, which is called NO BEE for short, performs an extensive search in the full feasible region of function $\gamma_G(u, v)$ (equation 3.2) and uses the full capacity of the GPU (described in section 4.1) by using as much work-items as necessary. The second program, called BEE, is the implementation of the Parallel Honeybee Search Algorithm (more details in section 3.4) using function $\gamma_G(u, v)$ as fitness function; the number of work-items used in BEE is bounded by the maximum size of the work-group: 256 work-items. As detailed in section 3.4, the Parallel Honeybee Search Algorithm requires synchronization between work-items to perform effectively, and this is only possible when using a single CU (Compute Unit), this is why only one work-group of the mentioned size is possible.

Program BEE has to define a value for q to divide the $p = 256$ work-items available in groups that represent each individual. The selected size of these groups is $q = 4$, meaning only 64 individuals are available to work simultaneously. This also means that the configuration parameters for the size of populations are all $\mu_e = \mu_h = \lambda_e = \lambda_h = 64$. The number 4 was selected since most of the objects of interest have a square shape (rather than a rectangle) and dividing a square in 4 pieces allows the even distribution of the work load.

The number of generations for the exploration phase was set to be 6. The

number of generations was selected during initial trials; more generations did not produce individuals with greater fitness. The same parameter for the harvest phase is set to be half of the number used for exploration because of the search space reduction (explained in section 3.3), in this case, 3 generations.

Other parameters for BEE are set using the values that gave good results in the experiments performed by Olague and Puente (2006a) in the original proposal of the sequential version of the algorithm. Those parameters are mainly related to the generation of offspring in the exploration and harvest phases. The values used for Polynomial Mutation and SBX are $\eta_m = 25$ and $\eta_c = 2$, respectively. The sizes of the subdivisions of offspring (λ) populations are expressed as percentages rather than static numbers. The proportions of sons generated through mutation are $\alpha_e/\lambda_e = 0.6$ and $\alpha_r/\lambda_r = 0.6$. The sons generated using crossover are $\beta_e/\lambda_e = 0.1$ and $\beta_r/\lambda_r = 0.3$. And the random offspring proportions are $\gamma_e/\lambda_e = 0.3$ and $\gamma_r/\lambda_r = 0.1$.

Both, BEE and NO BEE require the parameter `max_window` to be defined in order to reduce the size of images (as illustrated in Figure 3.5). This is done to avoid hitting the global memory size limit of the GPU, it also has the effect of reducing the size of the feasible region and thus the time cost of exploring it. The selected default value was set as `max_window` $= 24$, which was used for most of the tests, except for 3 specific video sequences where the size reduction was not enough. The image size of these 3 videos was too big, even when reduced, because the object of interest was small in comparison with the full frame:

1. The 14th video of the Motion Smoothness category (`max_window` of 12).
2. The 22th video of the Moving Camera category (`max_window` of 12).

3. The 2nd video of the Long Duration category (`max_window` of 8).

The data that is transmitted from the CPU to the GPU includes 3 image frames, the one where the image template t is already located, the preceding frame where t should also be located, and the one to be searched (f). Two image templates are used to improve the chances of finding the object of interest after a sudden change that is reversed in the next frame. The CPU also sends arrays of random numbers, video sequence specific values for m, n, u, v, w, h and all the configuration parameters that have been discussed in previous paragraphs (which can also be reviewed in Table 4.2).

Table 4.2: Summary of the configuration parameters used in the tests

Parameter	**BEE**	**NO BEE**
`max_window`	24 (most)	24 (most)
p (work-items)	256	–
q (work-items per bee)	4	–
μ_e (size of all populations)	64	–
Number of generations	6	–
η_m (parameter for mutation)	25	–
η_c (parameter for crossover)	2	–
α_e/λ_e (mutation sons, exploration)	60%	–
β_e/λ_e (crossover sons, exploration)	10%	–
γ_e/λ_e (random sons, exploration)	30%	–
α_r/λ_r (mutation sons, harvest)	60%	–
β_r/λ_r (crossover sons, harvest)	30%	–
γ_r/λ_r (random sons, harvest)	10%	–

4.4 Experimental Results

Two variables were measured in the experiments, time per frame and F-score for each video sequence. The description of the F-score metric can be found in section 3.5. Time was measured by registering the current time of the internal CPU clock when each frame processing began and finished. The time that was measured includes the time that was used to generate random numbers in the CPU and the time taken to send data from the CPU to the GPU.

The results show that there is not an important difference in the accuracy of the results between BEE and NO BEE, meaning the Parallel Honeybee Search Algorithm was able to find good results without exploring the whole feasible region. The survival curves for both versions of the algorithm can be seen in the upper section of Figure 4.4. The mean and standard deviation of the F-score for both programs is shown in Table 4.3, the same data is shown graphically in the bottom part of Figure 4.4. The difference between the means of BEE and NO BEE is of 0.011229303, this is equivalent to 3.66% of the standard deviation of both categories and thus is considered a trivial difference.

Table 4.3: Comparison of F-score between BEE and NO BEE

F-score

	Mean	**Standard deviation**
NO BEE	0.470193252	0.310460954
BEE	0.481422554	0.303214638
BEE and NO BEE	0.475807903	0.306665869

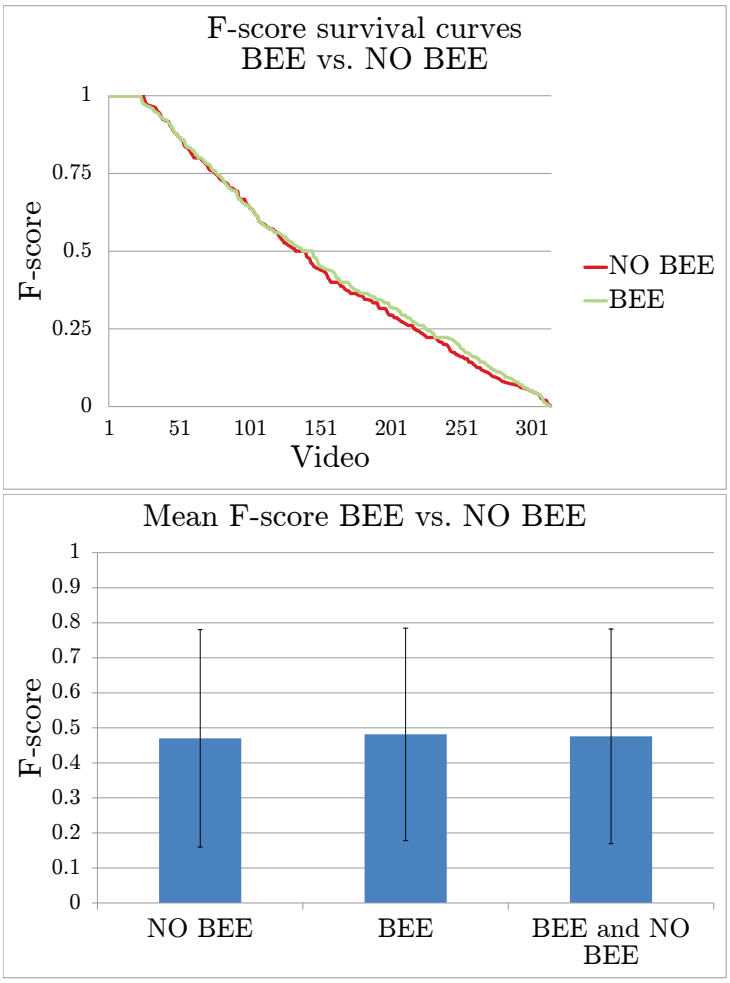

Figure 4.4: Comparison of F-score between BEE and NO BEE. Top plot shows comparison of F-score survival curves, bottom plot shows comparison of mean and standard deviation. There is not a significant difference between BEE and NO BEE in accuracy.

A more noticeable difference was detected by measuring the time taken by both programs to process each frame. BEE provides a more stable time, since it is not affected by the variability of the size of the search space. It is

also surprising that the version that implements the Parallel Honeybee Search Algorithm was able to be faster in average, even though less work-items are used (as explained in section 3.2.2). This difference in time can be observed in the top part of Figure 4.5, which shows the average time per frame of each video in the dataset for both versions. The averages are also shown in the figure, version NO BEE has an average of 0.179133 seconds per frame while version BEE has an average of 0.107322545 seconds per frame. The overall averages and standard deviations for time per frame when using each program can be found in Table 4.4, the same data is shown in the bottom part of Figure 4.5. The difference in the average time per frame between BEE and NO BEE is of 0.071810455 seconds, this is 38.06% of the standard deviation of both BEE and NO BEE. The difference in the average time may not be so noticeable between both groups; but the difference of the standard deviation between them is notable. The standard deviation of NO BEE is 7 times greater than the standard deviation of BEE.

Table 4.4: Comparison of time per frame between BEE and NO BEE
Time per frame

	Mean	**Standard deviation**
NO BEE	0.179133	0.259507849
BEE	0.107322545	0.036998953
BEE and NO BEE	0.143227772	0.188661255

The accuracy of BEE and NO BEE can be compared with the available data of other algorithms that were tested with the ALOV++ dataset. Figure 4.6 shows how BEE compares in accuracy against other 19 video tracking algorithms, see section 1.1.1 for brief mentions of the algorithms. Since

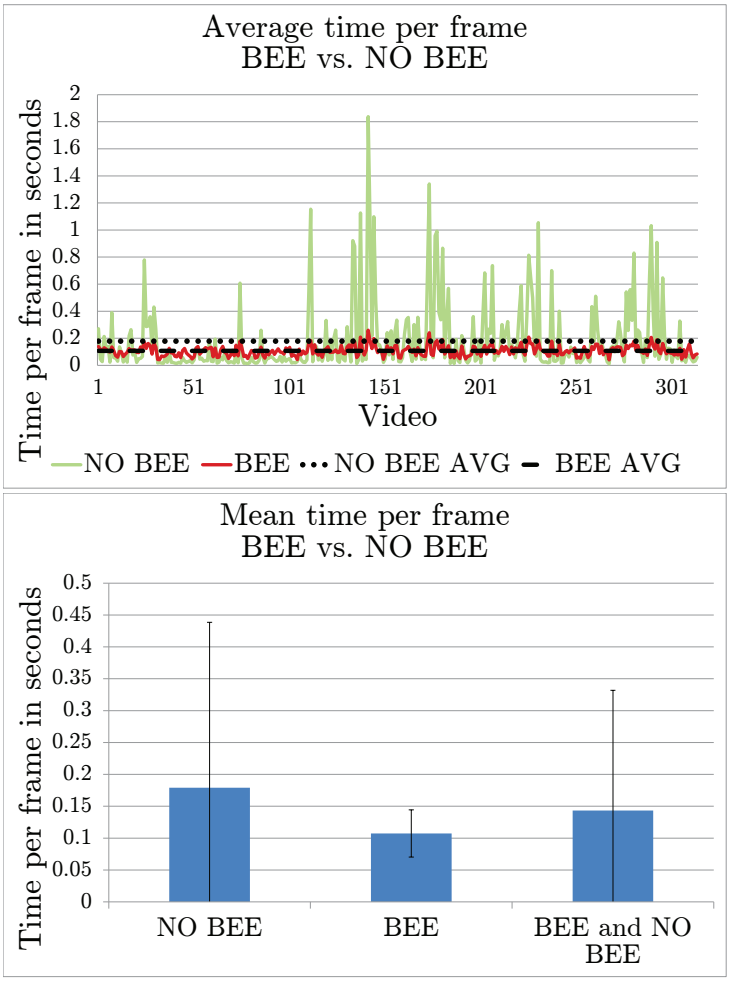

Figure 4.5: Comparison of time per frame between BEE and NO BEE. Top plot shows comparison of time per frame for each video, bottom plot shows comparison of mean and standard deviation. There is a significant difference between BEE and NO BEE in time per frame.

the illustration shows too much data and can be hard to read, NO BEE is excluded and assumed to behave very similarly to BEE (as can be seen in Figure 4.4). If these 21 algorithms are ordered by average F-score, BEE falls

in the 13th place and NO BEE in the 14th (see Table 4.5).

Table 4.5: Video tracking algorithms ordered by average F-score. The labels used for every algorithm can be found in Table 3.2.

	Avg. F-score	**Algorithm**		**Avg. F-score**	**Algorithm**
1	0.655044788	STR	11	0.515131077	FRT
2	0.646832576	FBT	12	0.48655316	HBT
3	0.6209639	TST	13	0.481422554	BEE
4	0.607268742	TLD	14	0.470193252	NO BEE
5	0.558174956	MIT	15	0.466671402	SPT
6	0.557619335	NCC	16	0.4570484	LKT
7	0.548627419	L1O	17	0.418726	KAT
8	0.548396965	L1T	18	0.353297697	MST
9	0.522182931	IVT	19	0.336014071	TAG
10	0.516015278	LOT	20	0.269204852	ACT
			21	0.147033336	TMC

It is surprising that BEE and NCC (6th place) are so far in the rank, since BEE is based on a similar algorithm (ZNCC). The fact that BEE is worse in accuracy can be attributed to the loss of information caused by the resizing of the image (procedure is described in section 3.2.2) and the combination of Sobel filter and ZNCC for fitness function. In any case, this cannot be attributed the Parallel Honeybee Search Algorithm, since the F-score results are almost identical to NO BEE, which does not use the Parallel Honeybee Search Algorithm but does resize the image frame due to the restraints in global memory size of the GPU (see section 4.1) and use the same fitness function.

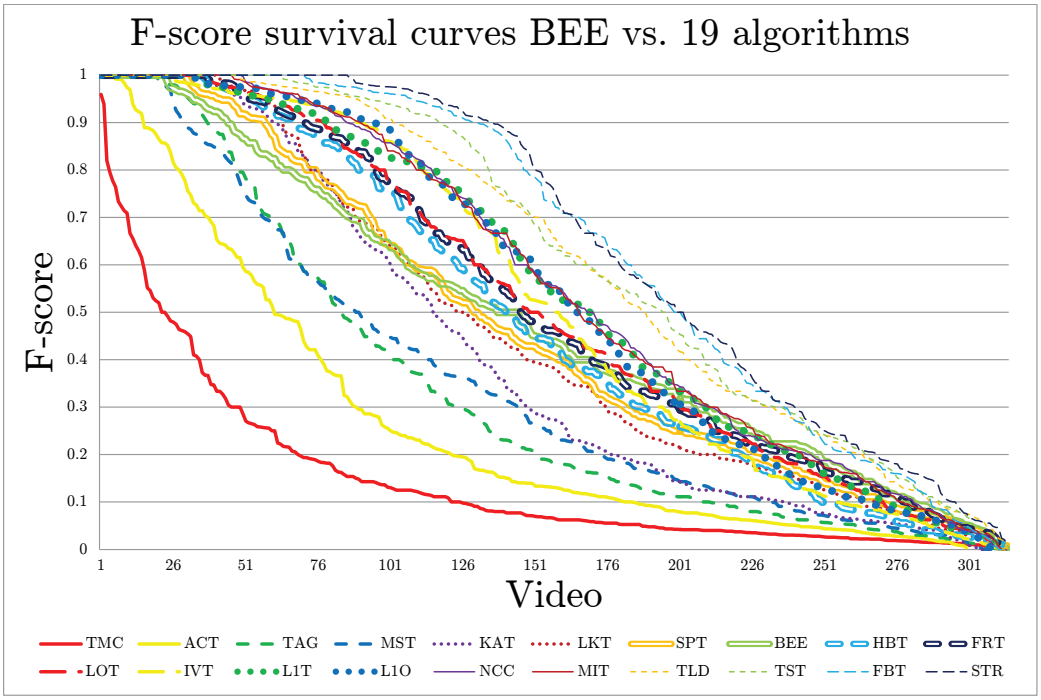

Figure 4.6: F-score survival curves of BEE and other 19 algorithms. Information obtained from Smeulders et al. (2014). The labels used for every algorithm can be found in Table 3.2.

Conclusions

The main objective of the present thesis was to analyze, design and implement a parallel version of the Honeybee Search Algorithm to be used for video tracking; and to be tested with a Graphics Processing Unit (GPU). The main contribution is the development of said implementation and the demonstration that the Parallel Honeybee Search Algorithm can successfully be used to improve the time cost of a video tracking algorithm in given situations, with little effect on the accuracy of the results.

To do this, a combination of the Zero Mean Normalized Cross-Correlation (ZNCC) and the Sobel filter was developed and parallelized to serve as base video tracking component (labeled as NO BEE for short). Then, the Honeybee Search Algorithm was adapted for parallelization and implemented using the same base function (labeled BEE for short). Later, the accuracy of BEE and NO BEE was tested and compared against each other and other algorithms.

The most important conclusions that can be drawn of this thesis are the following:

- The Parallel Honeybee Search Algorithm can be used to noticeably decrease the time taken by a given video tracking algorithm to find an object in a given image frame of considerable size (section 4.4). As the

size of the images in the video sequence increases, a given video tracking algorithm has to perform a broader search to provide results, but if the Parallel Honeybee Search Algorithm is used to conduct the exploration of the possible answers, time becomes less dependent on the size of the image frame and stabilizes because a smaller number of possibilities are reviewed. This conclusion was drawn from the experiments where programs BEE (using the Parallel Honeybee Search Algorithm) and NO BEE (not using the Parallel Honeybee Search Algorithm) were compared (more details in section 4.4).

- The Parallel Honeybee Search Algorithm can be used together with a given video tracking algorithm and preserve its accuracy to find an object in a given image frame (Section 4.4). Even though the Parallel Honeybee Search Algorithm checks a lesser amount of the possible answers that can be given as final result for a certain video tracking algorithm, it converges to the actual optimal solution and in most cases delivers results as accurate as the ones obtained by an extensive search. The experiments described in section 4.4 provide evidence for this conclusion, based on the comparison of programs BEE and NO BEE.

- Reducing the size of the image frame before it is sent from the CPU to the GPU produces an unfavorable effect on the accuracy of a given video tracking algorithm. Although this kind of preprocessing helps to avoid memory limitations of common GPUs and decreases the time of communication, it should be avoided due to the strong negative effects

in the overall accuracy of the video tracking algorithm in question. The results of the experiments in section 4.4 shows the effects, the accuracy of BEE and NO BEE was compared against the accuracy of other 19 algorithms used for video tracking. Specifically, the comparison against the video tracking algorithm NCC, which is very similar to the base video tracking algorithm of BEE and NO BEE, shows that the accuracy is reduced as described.

Future Work

- **If the objective is to improve accuracy:**
  - **Use other video tracking algorithms.** ZNCC and the Sobel filter were selected for their simplicity and straight forward approach, in comparison with other video trackers (section 1.1.1). Still, the number of video tracking algorithms in the literature provides many alternatives that could be used in conjunction with The Parallel Honeybee Search Algorithm. Further experiments could be made to compare how the Parallel Honeybee Search Algorithm behaves with different video tracking functions. As explained in section 1.2.1, the Parallel Honeybee Search Algorithm has the advantage of being based on Population Based Optimization, this allows the flexibility to use practically any video tracking algorithm by simply using it as fitness function.
- **If the objective is to improve time:**
  - **Use other parallel computing technologies.** The roots of

the GPU are deeply related with image processing and it is one of the most commonly available parallel computing technologies, these are the reasons why it was selected for this research. But, it would be beneficial to observe the possibility of implementing the Parallel Honeybee Search Algorithm for video tracking using other technologies such as clusters and FPGAs (Field Programmable Gate Array). As explained in sections 3.4 and 3.2.2, The usage of the GPU in this research came with several limitations: tested image frames had to be reduced due to memory restraints, and there was a limit to the number of work-items that could be used if synchronization was needed. This limited the size of the population that was used in the experiments to 64 individuals (see section 4.3).

- **Use more than one video tracking algorithm.** The Parallel Honeybee Search Algorithm has two main phases that require an evaluation tool for each of the possible answers to the problem (section 1.2.3.1). This evaluation tools could be two different video tracking algorithms; one for the exploration phase that is less reliable but quicker could help to get a general vision of the search space; and a second one for the harvest phase that is more reliable. Once the behavior of the Parallel Honeybee Search Algorithm with different video tracking algorithms could be established, the options could be evaluated and later implemented for experimentation.
- **Experiments with the multiswarm parallel model.** When using a GPU to parallelize Swarm Intelligence algorithms, such as

the Honeybee Search Algorithm, there are several parallel models to be considered (described in section 2.2). The parallel model used in the experiments of this thesis is the all-GPU parallel model, since all the Swarm Intelligence operations are performed in the GPU. Synchronization is a big issue in GPUs (as described in section 4.1), CUs (Computing Units) in a GPU have no contact with each other, but since coordination is possible between their own work-items it is possible to implement one swarm per CU (multiswarm). Further experiments with the described model may allow the usage of the full capabilities of the GPU with relatively small modifications to the Parallel Honeybee Search Algorithm.

– **Benchmarking against real-time video tracking.** The comparisons that were performed in section 4.4 against other algorithms were mainly concerned about accuracy. This is because the ALOV++ dataset (Smeulders et al., 2014) does not provide information about the time per frame of each algorithm that was tested. Since time is not reported and there is no mention about parallel computing, the implementations are assumed to be sequential and not concerned with real-time video tracking. Other benchmarking tools focused on real-time video tracking should be found and evaluated using the Parallel Honeybee Search Algorithm in search for more significant results.

References

Abbass, H. A. (2001). A single queen single worker honey–bees approach to 3-SAT. In *Proceedings of the 3rd annual conference on genetic and evolutionary computation* (pp. 807–814).

Adam, A., Rivlin, E., & Shimshoni, I. (2006). Robust fragments-based tracking using the integral histogram. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on* (Vol. 1, pp. 798–805).

Babenko, B., Yang, M.-H., & Belongie, S. (2009). Visual tracking with online multiple instance learning. In *Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on* (pp. 983–990).

Baker, S., & Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, *56*(3), 221–255.

Bätz, M., Richter, T., Garbas, J.-U., Papst, A., Seiler, J., & Kaup, A. (2014). High dynamic range video reconstruction from a stereo camera setup. *Signal Processing: Image Communication*, *29*(2), 191–202.

Bitam, S., Batouche, M., & Talbi, E. (2010). A survey on bee colony algorithms. *2010 IEEE International Symposium on Parallel & Distributed Processing*, 1-8.

Blecic, I., Cecchini, A., & Trunfio, G. A. (2014). Fast and accurate optimization of a GPU-accelerated CA urban model through cooperative coevolutionary particle swarms. *Procedia Computer Science*, *29*, 1631–1643.

Boyd, J. E., Hushlak, G., & Jacob, C. J. (2004). SwarmArt: interactive art from swarm intelligence. In *Proceedings of the 12th annual ACM international conference on multimedia* (pp. 628–635).

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc.

Briechle, K., & Hanebeck, U. D. (2001). Template matching using fast normalized cross correlation. In *Proc. SPIE* (Vol. 4387, pp. 95–102).

Calazan, R. M., Nedjah, N., & de Macedo Mourelle, L. (2012). Swarm grid: a proposal for high performance of parallel particle swarm optimization using GPGPU. In *International conference on computational science and its applications* (pp. 148–160).

Calazan, R. M., Nedjah, N., & de Macedo Mourelle, L. (2013). Three alternatives for parallel GPU-based implementations of high performance particle swarm optimization. In *International work-conference on artificial neural networks* (pp. 241–252).

Cardenas-Montes, M., Vega-Rodriguez, M. A., Rodriguez-Vazquez, J. J., & Gomez-Iglesias, A. (2011). Accelerating particle swarm algorithm with GPGPU. In *Parallel, distributed and network-based processing (PDP), 2011 19th euromicro international conference on* (pp. 560–564).

Castillo, R. (2016). *Implementación del filtro de función discriminante sintética para el reconocimiento de objetos en el róbot humanoide nao*

(Unpublished master's thesis). Universidad Autónoma de San Luis Potosí.

Catanzaro, B. (2010). OpenCL optimization case study: Simple reductions. *White Paper*.

Čehovin, L., Kristan, M., & Leonardis, A. (2011). An adaptive coupled-layer visual model for robust visual tracking. In *Computer vision (ICCV), 2011 IEEE international conference on* (pp. 1363–1370).

Chandra, R. (2001). *Parallel programming in openmp*. Morgan kaufmann.

Collett, D. (2015). *Modelling survival data in medical research*. CRC press.

Comaniciu, D., Ramesh, V., & Meer, P. (2000). Real-time tracking of non-rigid objects using mean shift. In *Computer vision and pattern recognition, 2000. proceedings. IEEE conference on* (Vol. 2, pp. 142–149).

Crescenzi, P., & Kann, V. (1997). Approximation on the web: A compendium of NP optimization problems. In *International workshop on randomization and approximation techniques in computer science* (pp. 111–118).

Crespi, B. J., & Yanega, D. (1995). The definition of eusociality. *Behavioral Ecology*, *6*(1), 109–115.

Crist, E. (2004). Can an insect speak? the case of the honeybee dance language. *Social Studies of Science*, *34*(1), 7–43.

Davidson, A., Tarjan, D., Garland, M., & Owens, J. D. (2012). Efficient parallel merge sort for fixed and variable length keys. In *Innovative parallel computing (InPar), 2012* (pp. 1–9).

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons.

Deb, K., & Beyer, H. (1999). *Self-adaptive genetic algorithms with simulated binary crossover* (first ed.). Secretary of the SFB 531.

Dhaussy, P., Filloque, J.-M., Pottier, B., & Rubini, S. (1994). Global control synthesis for an mimd/fpga machine. In *Fpgas for custom computing machines, 1994. proceedings. ieee workshop on* (pp. 72–81).

Di Stefano, L., Mattoccia, S., & Tombari, F. (2005). ZNCC-based template matching using bounded partial correlation. *Pattern recognition letters*, *26*(14), 2129–2134.

Draper, B. A., Beveridge, J. R., Bohm, A. W., Ross, C., & Chawathe, M. (2003). Accelerated image processing on FPGAs. *IEEE transactions on image processing*, *12*(12), 1543–1551.

Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE transactions on computers*, *100*(9), 948–960.

Foster, I. (1995). *Designing and building parallel programs* (Vol. 78). Addison Wesley Publishing Company Boston.

Galoogahi, H. K., Fagg, A., Huang, C., Ramanan, D., & Lucey, S. (2017). Need for speed: A benchmark for higher frame rate object tracking. *arXiv preprint arXiv:1703.05884*.

Gaster, B., Howes, L., Kaeli, D. R., Mistry, P., & Schaa, D. (2012). *Heterogeneous computing with OpenCL: Revised OpenCL 1.* Newnes.

Gemmell, J., Toyama, K., Zitnick, C. L., Kang, T., & Seitz, S. (2000). Gaze awareness for video-conferencing: A software approach. *IEEE MultiMedia*, *7*(4), 26–35.

Godec, M., Roth, P. M., & Bischof, H. (2013). Hough-based tracking of non-rigid objects. *Computer Vision and Image Understanding*, *117*(10),

1245–1256.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning, 1989. *Reading: Addison-Wesley*.

Hare, S., Golodetz, S., Saffari, A., Vineet, V., Cheng, M.-M., Hicks, S. L., & Torr, P. H. (2016). Struck: Structured output tracking with kernels. *IEEE transactions on pattern analysis and machine intelligence*, *38*(10), 2096–2109.

Held, D., Thrun, S., & Savarese, S. (2016). Learning to track at 100 FPS with deep regression networks. In *European conference computer vision (ECCV)*.

Hii, A., Hann, C. E., Chase, J. G., & Van Houten, E. E. (2006). Fast normalized cross correlation for motion tracking using basis functions. *Computer methods and programs in biomedicine*, *82*(2), 144–156.

Hoshino, J., Yamamoto, M., & Saito, H. (2001). A match moving technique for merging cg cloth and human movie sequences. *Computer Animation and Virtual Worlds*, *12*(1), 23–29.

Hsieh, H.-T., & Chu, C.-H. (2011). Particle swarm optimisation (PSO)-based tool path planning for 5-axis flank milling accelerated by graphics processing unit (GPU). *International Journal of Computer Integrated Manufacturing*, *24*(7), 676–687.

Hwang, K., Dongarra, J., & Fox, G. C. (2013). *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann.

Juneja, M., & Sandhu, P. S. (2009). Performance evaluation of edge detection techniques for images in spatial domain. *international journal of*

computer theory and Engineering, *1*(5), 614.

Kalal, Z., Matas, J., & Mikolajczyk, K. (2010). PN learning: Bootstrapping binary classifiers by structural constraints. In *Computer vision and pattern recognition (cvpr), 2010 IEEE conference on* (pp. 49–56).

Kalivarapu, V., & Winer, E. (2008). Implementation of digital pheromones in PSO accelerated by commodity graphics hardware. In *12th AIAA/ISSMO multidisciplinary analysis and optimization conference, victoria, british columbia.*

Kamakura, M. (2011). Royalactin induces queen differentiation in honeybees. *Nature*, *473*(7348), 478–483.

Kapinchev, K., Bradu, A., Barnes, F., & Podoleanu, A. (2015). GPU implementation of cross-correlation for image generation in real time. In *Signal processing and communication systems (icspcs), 2015 9th international conference on* (pp. 1–6).

Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization* (Tech. Rep.). Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Karaboga, D., & Akay, B. (2009). A survey: Algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, *31*(1-4), 61–85.

Kaufman, K. R., Hughes, C., Morrey, B. F., Morrey, M., & An, K.-N. (2001). Gait characteristics of patients with knee osteoarthritis. *Journal of biomechanics*, *34*(7), 907–915.

Khare, V., Yao, X., & Deb, K. (2003). Performance scaling of multi-objective evolutionary algorithms. In *Evolutionary multi-criterion optimization* (pp. 72–72).

Kwon, J., & Lee, K. M. (2009). Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive basin hopping monte carlo sampling. In *Computer vision and pattern recognition, 2009. cvpr 2009. IEEE conference on* (pp. 1208–1215).

Kwon, J., Lee, K. M., & Park, F. C. (2009). Visual tracking via geometric particle filtering on the affine group with optimal importance functions. In *Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on* (pp. 991–998).

Lawless, J. F. (2011). *Statistical models and methods for lifetime data* (Vol. 362). John Wiley & Sons.

Lewis, J. P. (1995). Fast normalized cross-correlation. In *Vision interface* (Vol. 10, pp. 120–123).

Li, M., & Lavest, J.-M. (1996). Some aspects of zoom lens camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *18*(11), 1105–1110.

Litvin, A., Konrad, J., & Karl, W. C. (2003). Probabilistic video stabilization using kalman filtering and mosaicing. In *Proceedings of spie* (Vol. 5022, pp. 663–674).

Lucic, P., & Teodorovic, D. (2001). Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. In *Preprints of the tristan iv triennial symposium on transportation analysis* (pp. 441–445).

Luenberger, D. G., Ye, Y., et al. (1984). *Linear and nonlinear programming* (Vol. 2). Springer.

Luo, G.-H., Huang, S.-K., Chang, Y.-S., & Yuan, S.-M. (2014). A parallel bees

algorithm implementation on GPU. *Journal of Systems Architecture*, *60*(3), 271–279.

Maggio, E., & Cavallaro, A. (2011). *Video tracking: theory and practice*. John Wiley & Sons.

Maia, R. D., de Castro, L. N., & Caminhas, W. M. (2012). Bee colonies as model for multimodal continuous optimization: The OptBees algorithm. In *Evolutionary computation (CEC), 2012 IEEE congress on* (pp. 1–8).

Malamas, E. N., Petrakis, E. G., Zervakis, M., Petit, L., & Legat, J.-D. (2003). A survey on industrial vision systems, applications and tools. *Image and vision computing*, *21*(2), 171–188.

Mantor, M. (2012). AMD Radeon HD 7970 with graphics core next (GCN) architecture. In *Hot Chips 24 Symposium (HCS), 2012 IEEE* (pp. 1–35).

Maurer, T., Elagin, E. V., Nocera, L. P. A., Steffens, J. B., & Neven, H. (2001, August 7). *Wavelet-based facial motion capture for avatar animation*. Google Patents. (US Patent 6,272,231)

Mei, X., & Ling, H. (2009). Robust visual tracking using ℓ_1 minimization. In *Computer vision, 2009 IEEE 12th international conference on* (pp. 1436–1443).

Mei, X., Ling, H., Wu, Y., Blasch, E., & Bai, L. (2011). *Minimum error bounded efficient l1 tracker with occlusion detection (preprint)* (Tech. Rep.). AIR FORCE RESEARCH LAB WRIGHT-PATTERSON AFB OH.

Mohanapriya, D., & Mahesh, K. (2017). A comparative analysis of video tracking techniques. *extraction*, *5*, 8.

Mozaffari, A., Gorji-Bandpy, M., & Gorji, T. B. (2012). Optimal design of constraint engineering systems: application of mutable smart bee algorithm. *International Journal of Bio-Inspired Computation*, *4*(3), 167–180.

Mussi, L., Nashed, Y. S., & Cagnoni, S. (2011). GPU-based asynchronous particle swarm optimization. In *Proceedings of the 13th annual conference on genetic and evolutionary computation* (pp. 1555–1562).

Nakrani, S., & Tovey, C. (2003). On honey bees and dynamic allocation in an internet server colony. In *Proceedings of 2nd international workshop on the mathematics and algorithms of social insects* (pp. 1–8).

Nguyen, H. T., & Smeulders, A. W. (2004). Fast occluded object tracking by a robust appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(8), 1099–1104.

Nguyen, H. T., & Smeulders, A. W. (2006). Robust tracking using foreground-background texture discrimination. *International Journal of Computer Vision*, *69*(3), 277–293.

Nickolls, J., Buck, I., Garland, M., & Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, *6*(2), 40–53.

Nothdurft, T., Hecker, P., Ohl, S., Saust, F., Maurer, M., Reschka, A., & Böhmer, J. R. (2011). Stadtpilot: First fully autonomous test drives in urban traffic. In *Intelligent transportation systems (ITSC), 2011 14th international IEEE conference on* (pp. 919–924).

Nowak, M. A., Tarnita, C. E., & Wilson, E. O. (2010). The evolution of eusociality. *Nature*, *466*(7310), 1057–1062.

Ntuen, C. A., Park, E. H., & Kim, J. H. (1989). KIMSa knowledge-based

computer vision system for production line inspection. *Computers & Industrial Engineering*, *16*(4), 491–508.

Olague, G., & Puente, C. (2006a). The honeybee search algorithm for three-dimensional reconstruction. In *Workshops on applications of evolutionary computation* (pp. 427–437).

Olague, G., & Puente, C. (2006b). Parisian evolution with honeybees for three-dimensional reconstruction. In *Proceedings of the 8th annual conference on genetic and evolutionary computation* (pp. 191–198).

Oron, S., Bar-Hillel, A., Levi, D., & Avidan, S. (2015). Locally orderless tracking. *International Journal of Computer Vision*, *111*(2), 213–228.

Owaida, M., Bellas, N., Daloukas, K., & Antonopoulos, C. D. (2011). Synthesis of platform architectures from OpenCL programs. In *Field-programmable custom computing machines (FCCM), 2011 IEEE 19th annual international symposium on* (pp. 186–193).

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). Gpu computing. *Proceedings of the IEEE*, *96*(5), 879–899.

Patnaik, S., & Yang, Y.-M. (2012). *Soft computing techniques in vision science* (Vol. 395). Springer Science & Business Media.

Patten, J., Ishii, H., Hines, J., & Pangaro, G. (2001). Sensetable: a wireless object tracking platform for tangible user interfaces. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 253–260).

Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. (2011). The bees algorithm-a novel tool for complex optimisation. In *Intelligent production machines and systems-2nd I* PROMS virtual international*

conference (3-14 july 2006).

Prabhu, C. (2008). *Grid and cluster computing.* PHI Learning Pvt. Ltd.

Quijano, N., & Passino, K. M. (2010). Honey bee social foraging algorithms for resource allocation: Theory and application. *Engineering Applications of Artificial Intelligence*, *23*(6), 845–861.

Raimbault, F., Lavenier, D., Rubini, S., & Pottier, B. (1993). Fine grain parallelism on a mimd machine using fpgas. In *Fpgas for custom computing machines, 1993. proceedings. ieee workshop on* (pp. 2–8).

Rao, G. N., Rao, P. J., Duvvuru, R., Bendalam, S., & Gemechu, R. (2016). An enhanced real-time forest fire assessment algorithm based on video by using texture analysis. *Perspectives in Science*, *8*, 618–620.

Ratha, N. K., Jain, A. K., & Rover, D. T. (1995). Convolution on SPLASH 2. In *FPGAs for custom computing machines, 1995. proceedings. IEEE symposium on* (pp. 204–213).

Ross, D. A., Lim, J., Lin, R.-S., & Yang, M.-H. (2008). Incremental learning for robust visual tracking. *International Journal of Computer Vision*, *77*(1), 125–141.

Schlüns, H., Koeniger, G., Koeniger, N., & Moritz, R. F. (2004). Sperm utilization pattern in the honeybee (Apis mellifera). *Behavioral Ecology and Sociobiology*, *56*(5), 458–463.

Senior, A. W., Brown, L., Hampapur, A., Shu, C.-F., Zhai, Y., Feris, R. S., ... Carlson, C. (2007). Video analytics for retail. In *Advanced video and signal based surveillance, 2007. AVSS 2007. IEEE conference on* (pp. 423–428).

Shrivakshan, G., Chandrasekar, C., et al. (2012). A comparison of various

edge detection techniques used in image processing. *IJCSI International Journal of Computer Science Issues*, *9*(5), 272–276.

Slessor, K. N., Winston, M. L., & Le Conte, Y. (2005). Pheromone communication in the honeybee (Apis mellifera L.). *Journal of chemical ecology*, *31*(11), 2731–2745.

Smeulders, A. W., Chu, D. M., Cucchiara, R., Calderara, S., Dehghan, A., & Shah, M. (2014). Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(7), 1442–1468.

Sobel, I., & Feldman, G. (1968). A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, 271–272.

Solihin, Y. (2015). *Fundamentals of parallel multicore architecture*. CRC Press.

Souza, D. L., Teixeira, O. N., Monteiro, D. C., & de Oliveira, R. C. L. (2012). A new cooperative evolutionary multi-swarm optimizer algorithm based on cuda parallel architecture applied to solve engineering optimization problems. In *3rdinternational workshop on combinations of intelligent methods and applications (CIMA 2012)* (p. 49).

Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, *12*(3), 66–73.

Tan, Y., & Ding, K. (2016). A survey on GPU-based implementation of swarm intelligence algorithms. *IEEE transactions on cybernetics*, *46*(9), 2028–2041.

Teodorovic, D., & Dell'Orco, M. (2005). Bee colony optimization–a coopera-

tive learning approach to complex transportation problems. *Advanced OR and AI methods in transportation*, 51–60.

Tian, Y.-l., Brown, L., Hampapur, A., Lu, M., Senior, A., & Shu, C.-f. (2008). IBM smart surveillance system (S3): event based video surveillance system with an open and extensible framework. *Machine Vision and Applications*, *19*(5), 315–327.

Trucco, E., & Plakas, K. (2006). Video tracking: a concise survey. *IEEE Journal of Oceanic Engineering*, *31*(2), 520–529.

Trucco, E., & Verri, A. (1998). *Introductory techniques for 3-D computer vision* (Vol. 201). Prentice Hall Englewood Cliffs.

Tsutsui, S., & Fujimoto, N. (2011). ACO with tabu search on a GPU for solving QAPs using move-cost adjusted thread assignment. In *Proceedings of the 13th annual conference on genetic and evolutionary computation* (pp. 1547–1554).

Tucker, K. W. (1957). *Automictic parthenogenesis in the honey bee, Apis mellifera L.* University of California, Davis.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–460.

Van Houten, E. E., Kershaw, H., Lotz, T., & Chase, J. G. (2012). Localization and detection of breast cancer tumors with digital image elastotomography. In *Engineering in medicine and biology society (embc), 2012 annual international conference of the ieee* (pp. 2635–2638).

Von Frisch, K. (1955). *The dancing bees*. Harcourt, Brace.

Von Frisch, K. (2014). *Bees: their vision, chemical senses, and language*. Cornell University Press.

Wang, S., Lu, H., Yang, F., & Yang, M.-H. (2011). Superpixel tracking. In *Computer vision (ICCV), 2011 IEEE international conference on* (pp. 1323–1330).

Wedde, H. F., Farooq, M., Pannenbaecker, T., Vogel, B., Mueller, C., Meth, J., & Jeruschkat, R. (2005). BeeAdHoc: an energy efficient routing algorithm for mobile ad hoc networks inspired by bee behavior. In *Proceedings of the 7th annual conference on genetic and evolutionary computation* (pp. 153–160).

Wedde, H. F., Farooq, M., & Zhang, Y. (2004). BeeHive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. *Lecture notes in computer science*, *3172*, 83–94.

Wilson, E. (1975). *Sociobiology: The new synthesis*. The Belknap Press.

Xie, J., Khan, S., & Shah, M. (2008). Automatic tracking of Escherichia coli bacteria. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2008*, 824–832.

Yang, X.-S. (2005). Engineering optimizations via nature-inspired virtual bee algorithms. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, 317–323.

Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE multimedia*, *19*(2), 4–10.

Zhao, J., Wang, W., Pedrycz, W., & Tian, X. (2012). Online parameter optimization-based prediction for converter gas system by parallel strategies. *IEEE Transactions on Control Systems Technology*, *20*(3), 835–845.

Zhou, Y., & Tan, Y. (2009). GPU-based parallel particle swarm optimization.

In *Evolutionary computation, 2009. CEC'09. IEEE congress on* (pp. 1493–1500).