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

Design and evaluation of a parallel video tracking system based on honeybee swarm behavior

T E S I S

Que para obtener el grado de:

Maestro en Ingeniería de la Computación

Presenta:
Juan Alvaro de la Rosa Ipiña

Asesor:
Dr. Carlos Soubervielle Montalvo

Co-asesor:
Dr. Cesar Augusto Puente Montejano

San Luis Potosí, S. L. P. Marzo de 2026

UASLP
Universidad Autónoma de San Luis Potosí

FACULTAD DE INGENIERÍA

19 de junio de 2025

ING. JUAN ALVARO DE LA ROSA IPIÑA
P R E S E N T E.

En atención a su solicitud de Temario, presentada por los **Dres. Carlos Soubervielle Montalvo y César Augusto Puente Montejano,** *Asesor y Coasesor del Trabajo de 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 19 de junio del presente, fue aprobado el Temario propuesto:

TEMARIO:

"Diseño y evaluación de un sistema paralelo de seguimiento de objetos en vídeo basado en el comportamiento de enjambre de abejas"

I. Introducción
II. Fundamentos de Algoritmos Evolutivos
III. Enfoques y Soluciones del Problema del Seguimiento de Objetos en Video
IV. Desarrollo del Sistema Paralelo para Seguimiento de Objetos en Video
V. Experimentos y Resultados
VI. Conclusiones y Trabajo Futuro
Bibliografía

"MODOS ET CUNCTARUM RERUM MENSURAS AUDEBO"

A T E N T A M E N T E

DR. EMILIO JORGE GONZÁLEZ GALVÁN
DIRECTOR

UNIVERSIDAD AUTONOMA
DE SAN LUIS POTOSI
FACULTAD DE INGENIERIA
DIRECCION

Ccp. Archivo
-bvlg

www.uaslp.mx

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

Abstract

This research seeks to improve the parallel implementation of ZNCC as a video tracking algorithm to achieve a considerable improvement by adapting the bio-inspired HSA (Honeybee Search Algorithm) as video tracking algorithm, using a newly adapted ZNCC version as the fitness function. To achieve this, adaptations to the HSA algorithm will need to be proposed according to the video tracking problem (i.e., population management during the exploration, recruitment, and harvesting phases). The completed tasks form the basis for implementing the parallel video tracking system more efficiently using the heterogeneous CPU-GPU architecture.

Agradecimientos

Agradezco a mi familia y principalmente a mis padres quienes siempre me ha mostrado su apoyo tanto en este par de años de maestría como en toda mi trayectoria académica.

A mi asesor de tesis y co-asesor quienes me introdujeron en este proyecto y me apoyaron a sacar adelante esta tesis. A mis sinodales y miembros de comité de tesis por estar al pendiente y apoyarme dando su retroalimentación a lo largo de los avances de tesis que les presente. Y finalmente a la Universidad Autónoma de San Luis Potosi por recibirme y formarme durante todos estos años tanto de licenciatura como de maestría, y especialmente a todos los maestros que me acompañaron a los largo de esos años.

Content

List of figures

List of tables

Chapter 1

Introduction

Computer vision is an area of computing that has been subject of research in recent decades in an attempt to replicate the sense of vision within a computer so that it can process information from physical universe as humans and other living beings in the animal kingdom do. Computer vision is, therefore, a subfield of artificial intelligence that involves image and video processing for various purposes including finding and tracking an object through a video sequence.

1.1 The video tracking problem and applications

Video object tracking (VOT) is a task that consists of identifying the position of an object (or several objects) in each frame of a video sequence following its trajectory [4] and determining its behavior (especially its location) along the video. VOT problem is essentially a long-form image recognition problem because it requires identifying a specific objective on the several images (frames) that make up a video (Figure 1.1). There are different approaches that have been used to address and describe the VOT problem (which will be described in more detail in Chapter 3) but essentially this is its basic definition and that from which the others branch off.

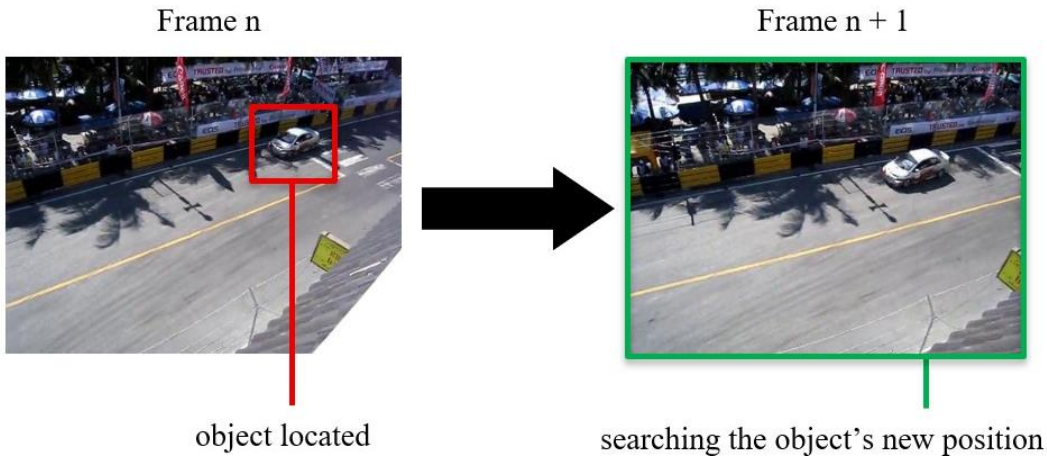

Figure 1.1. General video tracking process. The object located in a previous frame must be found on the actual frame.

VOT is one of the most requested tasks in the computer vision area due to its multiple application field with unlimited potential such as medical applications, military applications, entertainment, among others [1] [8]. Some of the applications that video track has include:

- Monitoring and analysis of body movement.
  - Rehabilitation control.
  - Video surveillance.
  - Motion capture for video games and cinema.
  - Sports analysis and transmission.
- General robotics.
  - Industrial robotics.
  - Medical robotics.
- Autonomous driving.
- Autonomous flying drones.
- Augmented reality.
- Virtual reality.
- Autonomous weapons.

For years, multiple video tracking algorithms and methodologies have been proposed in order to achieve this task with precision, however, the analysis of video frames is an exhaustive and delicate task that has not been perfected yet, in addition to entailing a

great demand for time and computational resources. For this reason, efforts have been made to enhance these systems through different software and hardware resources.

Improving object tracking methods is important due to the potential catastrophic failures that these could generate when applied to critical systems and especially to systems where the integrity of human lives is involved, such as medical, military, or certain everyday use systems, such as autonomous driving vehicles. These failures may be due to the susceptibility of video tracking algorithms to certain obstacles commonly present in video sequences such as light changes, object resize, similarity between objects, fast camera movements, low contrast, occlusion, etc., in addition to the tracking algorithms' large computational costs and complexity that cause considerably serious delays in response times.

This is why today video object tracking is a problem that still does not have a definitive solution, although different solutions and algorithms have been proposed, these remain unrefined and more research is required to find solutions that allow tracking systems to overcome the different obstacles that arise, including not only the noise and other problems present in the videos but also solving the optimization problems inherent to algorithms as complex as these that are designed to carry out a task as strenuous as video tracking, problems such as high consumption of energy and computing resources in general, which eventually cause extended response times by systems.

1.2 State of the art

Different VOT methods have been proposed over the years trying to find the best way to locate the position of a previously given object image within the current frame, however, none of the proposed methods have managed to solve the VOT problem 100%.

Some of the most recent and popular methods use machine learning and deep learning techniques, such as convolutional neural networks (SiamMask [6], SiamRPN++ [2], ByteTrack [7], etc.) and transformers (OSTrack [3], MixFormer [9], TransTrack [5], etc.). Although these methods have proven to be relatively effective, they also have major

performance disadvantages as they require training and, in some cases, large computational resources. This is why more traditional conventional methods such as region matching and image correlation are still used and studied.

One of the most well-known region matching methods is Zero Mean Normalized Cross-Correlation (ZNCC), a variant of the original NCC algorithm [42]. Some implementations of the ZNCC and NCC algorithms have been investigated, such as initial evaluations of this algorithm on the ALOV300++ dataset [11] where the NCC had an F-Score of 57% (Figure 1.2) compared to the highest one of 66% obtained by the Struck algorithm. In the VOT context, F-Score (and its most used variant: F1-Score) is a commonly used evaluation metric that represents the harmonic mean between Precision and Recall (but this will be explained in more detail in Chapter 4). In this previous NCC implementation, the NCC correlation filter was applied to 100% of the pixels in each frame of each video in the ALOV300++ dataset which resulted in considerable disadvantages in terms of optimization and execution time, which will be detailed later.

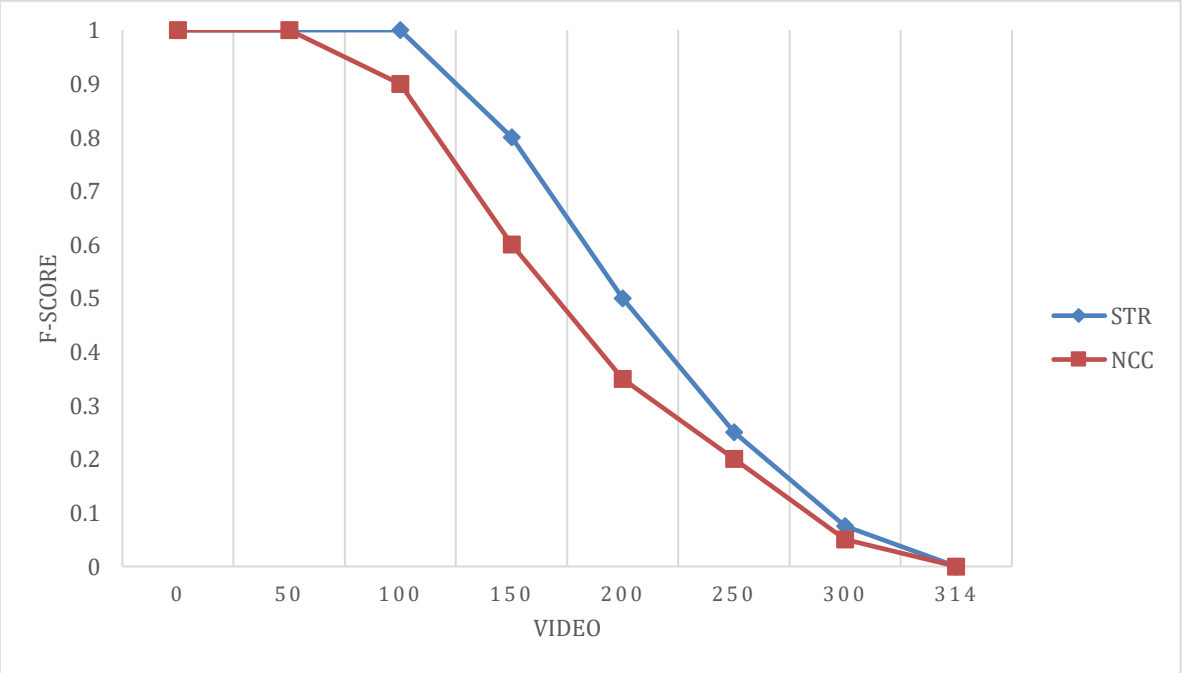

Figure 1.2. Survival curves from initial test with video tracking algorithms on ALOV300++ dataset [11]. As can be seen, Normalized Cross Correlation (NCC) got an F-Score of 57% against Struck's 66% F-Score (STR)

Into the VOT methods, bioinspired metaheuristics have been of special interest too. Algorithms inspired by nature have shown immense potential as optimal solutions capable of solving computational problems with high speed and low resource consumption. Some of the most popular bioinspired metaheuristics in recent years in video tracking research are ABC (Artificial Bee Colony) [12], BA (Bat Algorithm) [13] [14], and especially PSO (Particle Swarm Optimization) [8] which has several studies and papers published about how it can be used for video tracking.

Another bioinspired metaheuristic is the Honeybee Search Algorithm (HSA) [15] [16], proposed to be used in conjunction with the ZNCC algorithm as a fitness function. This implementation presents promising results (approximately 30% F1-Score and 13.8 FPS on the ALOV300++ dataset [17] [18]), but also presents areas of opportunity, such as the FPS rate, which is considerably low and leads to deeper investigations. This approach has not been touched in the literature to date, except for the works of Drs. Pérez-Cham, Puente and Soubervielle [15] [16] [17] [18] [19].

Among the algorithms with which this implementation is compared, Struck and SiamMask stand out. Struck is one of the most popular and well received online video tracking algorithms with the best results, while SiamMask is perhaps the most popular deep learning algorithm in the VOT area [17]. Against these two trackers, the NCC – HSA implementation obtained lower but also promising results (Figure 1.3) that indicated that after applying certain improvements it could be capable of obtaining comparable results.

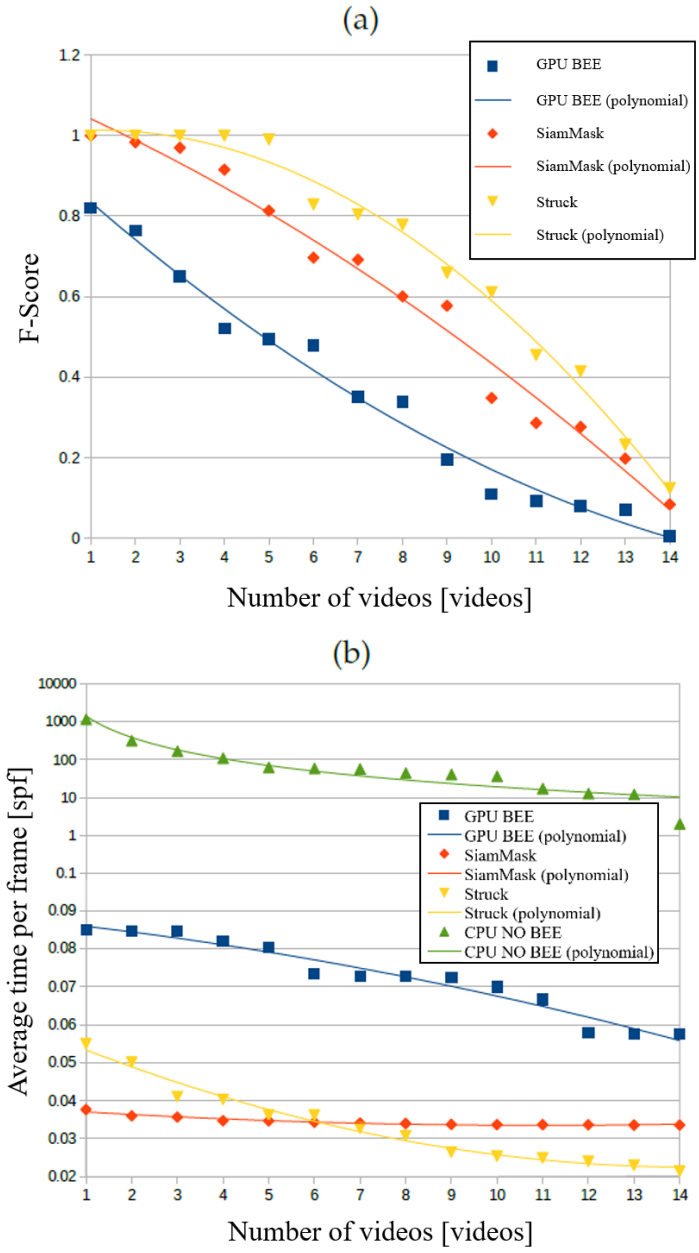

Figure 1.3. F-Score measurement and average time per frame comparison between the Dr. Pérez-Cham's NCC – HSA implementation vs. Struck and SiamMask algorithms [17]

This HSA-ZNCC implementation uses parallel computing, an approach that has been used in VOT for years to optimize and accelerate algorithms. Even if some parallel VOT proposals still use traditional CPUs to implement parallel solutions [20] [21], the emergence of increasingly specialized hardware architectures and components designed to

address specific computational tasks has driven the search for ways to harness their potential in current areas of computational research such as parallel computing itself.

An example of this is GPUs, whose architecture is specifically designed to process graphics tasks and can be powerful tools, particularly in applications such as VOT [22] [23] [24]. In addition to their potential in VOT algorithms (such as ZNCC), enormous potential has been observed for optimization and parallelization algorithms such as PSO [17] and HSA.

There are a lot of dataset benchmarks that have been used to evaluate VOT implementations such as the VOT Challenge benchmark [25], but one of the most used is the ALOV300++ benchmark [11] [26], a dataset composed of more than 300 videos divided into various categories according to the properties and obstacles they present such as occlusion, reflection, shape changes, long duration, etc. (Figure 1.4) After it was proposed in 2014, it became very popular in VOT research due to its diversity and easy access and even its use has been decreasing through time, it has been still used in recent years because of the previous reasons that were mentioned [17] [27].

The ALOV300++ dataset represents a challenge for any VOT system due to the robust diversity of its videos which allows us to observe in detail how a video tracker can have better or worst performance under certain conditions and certain types of videos, this is why this was the dataset chosen for this research. Due to the variety of challenges presented on the dataset, we have set F1-Score => 62% as goal to this project considering it would be a considerable improvement over the previous implementation [15], the initial results obtained by the individual implementation of the NCC algorithm [11], and it is somewhat close to the results obtained by Struck [11] without expectations of surpassing it. Also, a 62% F1-Score is far beyond the results that a VOT system could obtain by chance, especially on this dataset.

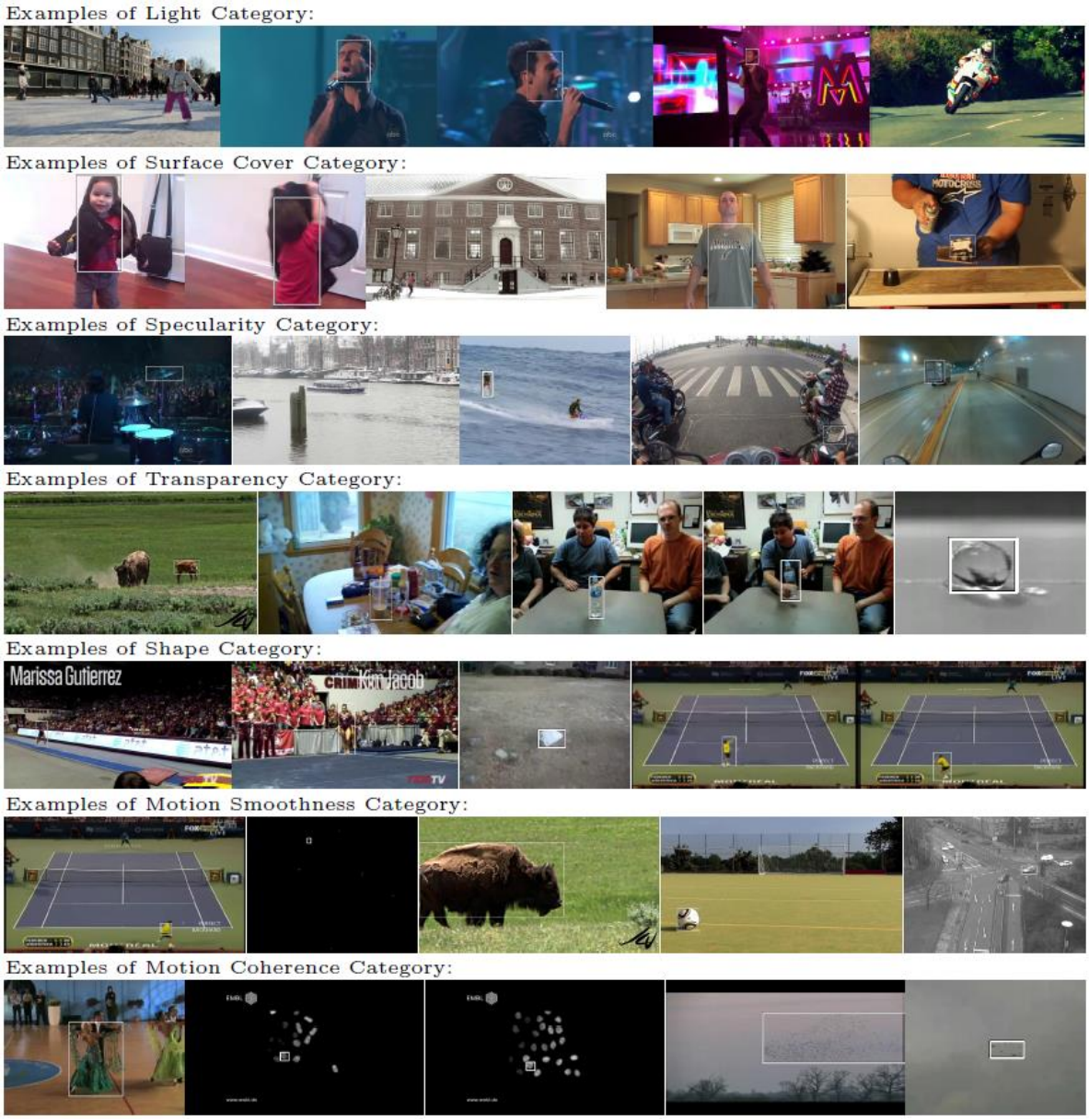

Figure 1.4. Video examples from some of the ALOV300++ categories [11]

1.3 Research questions

- What software adaptations should be made to increase F1-Score up to 62% of the HSA tracker using ZNCC as fitness function?

- How should a GPU-based video tracking system be developed to increase FPS by at least 15 without comprising its tracking capability?

1.4 Objectives

The main objective of this thesis is to design and implement a GPU-based video tracking system using Honeybee Search Algorithm (HSA) efficiently integrated with ZNCC method as fitness function to achieve up to 62% F1-Score using dataset benchmarks. In addition, a parallel video tracking system programmed in Python with a minimum speed of 15 frames per second (FPS) is expected.

The specific objectives of the thesis are:

1 Review and implement previous work related to GPU-based video tracking system programmed in C using HSA and ZNCC and test its performance using ALOV300++ dataset to find areas of opportunity.

2 Develop a new version of the GPU-based video tracking system programmed in Python using a component-based approach where the least possible coupling between its components is promoted, seeking a customizable version that considers usability and scalability.

3 Increase F1-Score up to 62% using the ALOV300++ dataset through continuous feedback of information generated in each frame of the video sequence, to optimize the performance of HSA and ZNCC in frame-by-frame object search.

4 Increase speed up to 15 FPS through correct distribution of HSA and ZNCC tasks taking advantage of Python programming and its GPU libraries.

5 Evaluate the system by implementing appropriate metrics such as F1-Score and FPS in the Python environment and consider dataset benchmarks such as ALOV300++.

1.5 Main contributions

- **Thesis:** in this thesis the contributions that the research will bring to the topic of VOT will be detailed and published. These advances are expected to be new both

individual and composed implementations of the ZNCC and HSA algorithms together for the optimization and improvement of the automated video tracking.

- **Component-based video tracking system implementation:** the main product of this research is a component-based code written in the Python language (with additions in C language) and composed of different software components in charge on different tasks such as the tracking itself, optimization, testing, etc., as well as a customizable and organized structure. This code is expected to be as open as possible, being available for use and modification by other researchers and individuals interested in its exploration and even expansion.
- **Code documentation:** along with the previously mentioned code, it is also expected to generate a kit with all the corresponding documentation including (but not limited to) a programmer's manual (including the documented code), user manual, among other documents and items.
- **Conference article:** an article has been already written and published for the MCPR (Mexican Conference on Pattern Recognition) Postgraduate Students' Meeting 2025 [10], an artificial intelligence research international conference.

1.6 Thesis organization

The next list describes the contents and the important points covered in the following chapters:

- **Chapter 2 – Foundations on Evolutionary Algorithms:** a general description of what evolutionary and swarm intelligence algorithms are, their classification, their elements (including the evolutionary operators) and a brief introduction to the Honeybee Search Algorithm and how it is used on VOT applications.
- **Chapter 3 – Approaches and Solutions of the Video Tracking Problem:** an introduction to the nature of the most well-known VOT proposals, emphasizing the ZNCC algorithm. Also we introduce the concept of parallelization through GPU (emphasizing its use on VOT) along with the VOT evaluation metrics.

- **Chapter 4 – Development of the Parallel Video Tracking System:** full description of the development of the VOT system specifying its component-based design (including distribution of parallel tasks), the proposed improvements and problems found.
- **Chapter 5** – **Experiments and results:** description of the different tests that were made to evaluate the system and the results obtained.
- **Chapter 6 – Conclusions and future work**: final discussion about the results described in the previous chapter and possible future work.

Chapter 2

Foundations on Evolutionary Algorithms

2.1 Evolutionary algorithms

Although evolutionary algorithms have been developed since the mid-1960s, the main concept was consolidated in the 1970s strongly inspired by the mechanism of biological evolution and the prevalence of the best individuals in each generation of a living population. Evolutionary algorithms were extensively researched during the following decades, during which different branches of this field emerged, such as genetic algorithms, evolutionary programming, genetic programming, and evolutionary strategies (Figure 2.1).

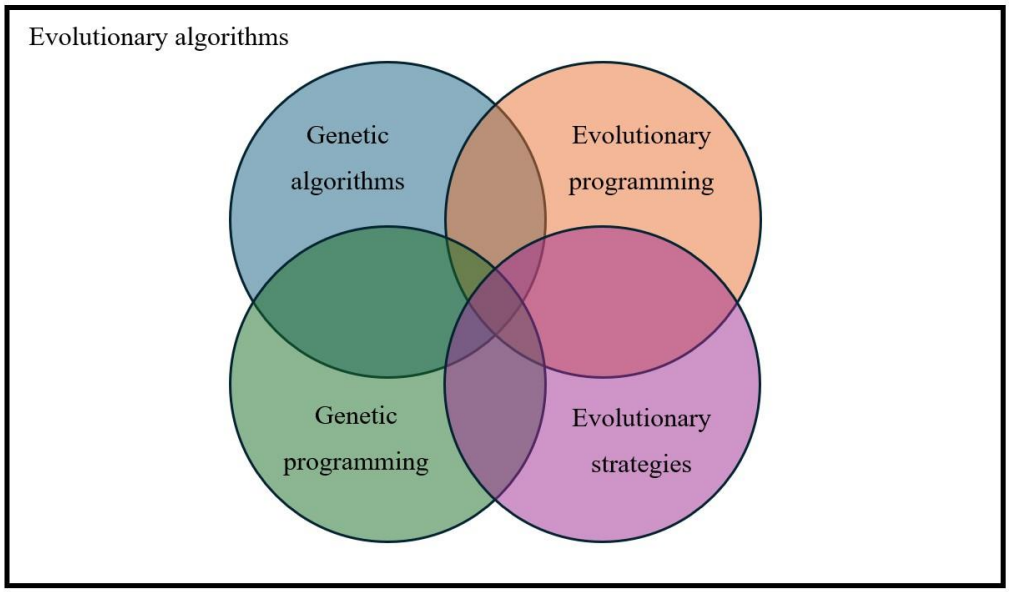

Figure 2.1. Genetic algorithms, evolutionary programming, genetic programming and evolutionary strategies are all branches of evolutionary algorithms which share common characteristics among themselves

In an evolutionary algorithm, we have a problem to solve, for which a set of proposed solutions would be equivalent to a population [28]. In nature, the fittest individuals reproduce by transmitting their best characteristics to their offspring, improving the species over generations. In evolutionary algorithms, this is equivalent to how each solution in the population is evaluated to find the best ones to carry out crossover and mutation processes through which the best characteristics are selected and used to generate a new population of solutions. This process is repeated for several generations until the best possible conditions for stopping the algorithm are found (Figure 2.2).

In evolutionary algorithms, a solution or individual within a population is known as "phenotype". It is the literal, tangible representation of a possible solution generated during the initialization stage of the evolutionary algorithm.

A "genotype" is the internal coding used in algorithms to represent phenotypes. It is an abstract representation of the possible solutions that compose the populations generated by the algorithm. Genotypes can range from simple bits and bytes to more complex data structures depending on the problem to be solved and the needs of the algorithm.

Within genotypes, "genes" (also called variables or locus) are the characteristics and elements that compose a genotype, which are the characteristics and properties of the solution or individual itself. In the selection and crossover process, the best individuals are selected and from these the best genes are taken to be passed to the next generation. In mutation process, genes are altered to change the individual's composition to bring them closer to the most optimal solution.

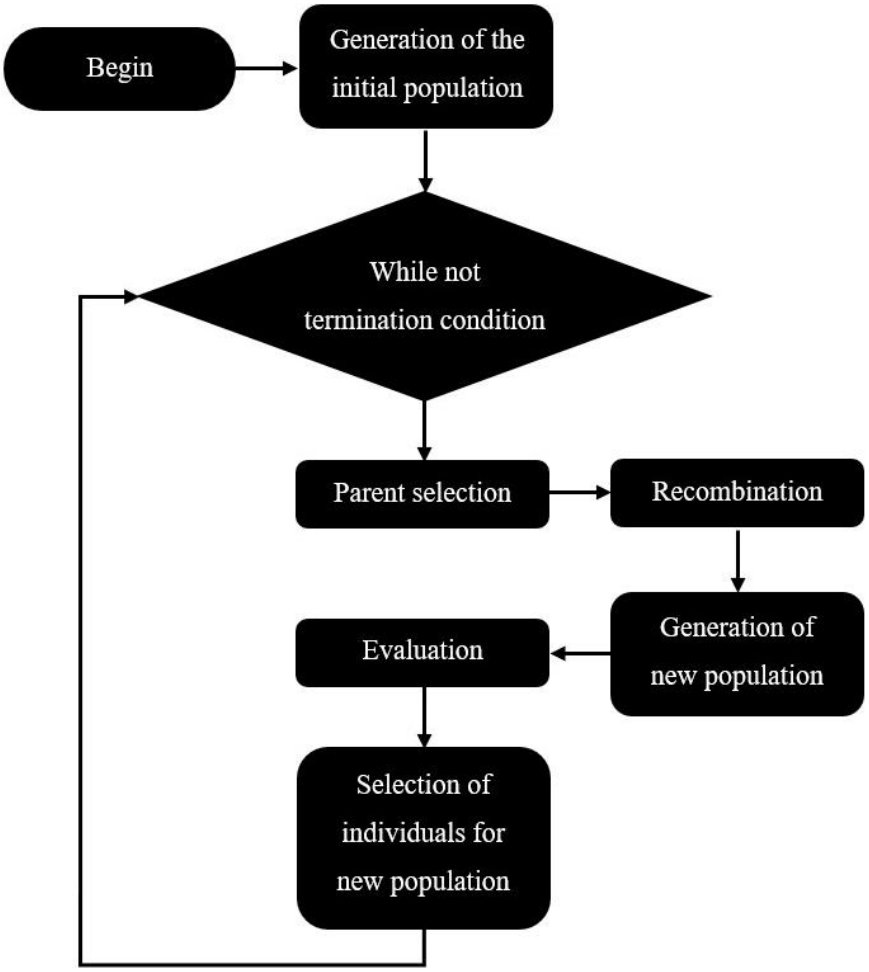

Figure 2.2. Flowchart describing a evolutionary algorithm in general terms

There are many methods that are used to carry out the four principal reproduction operators of the evolutionary algorithms: selection, crossover, mutation and copy.

In the selection phase, the algorithm chooses the best possible solutions to be recombined. The most popular selection methods are roulette and tournament [32] [35]. In the roulette method, candidates are chosen at random as if it were literally a roulette wheel, although a variation of this method known as "rigged roulette" is usually used where the values with the best fitness have a higher probability of being selected. In the tournament method, individuals are randomly paired to compete against each other until the best are chosen (Figure 2.3).

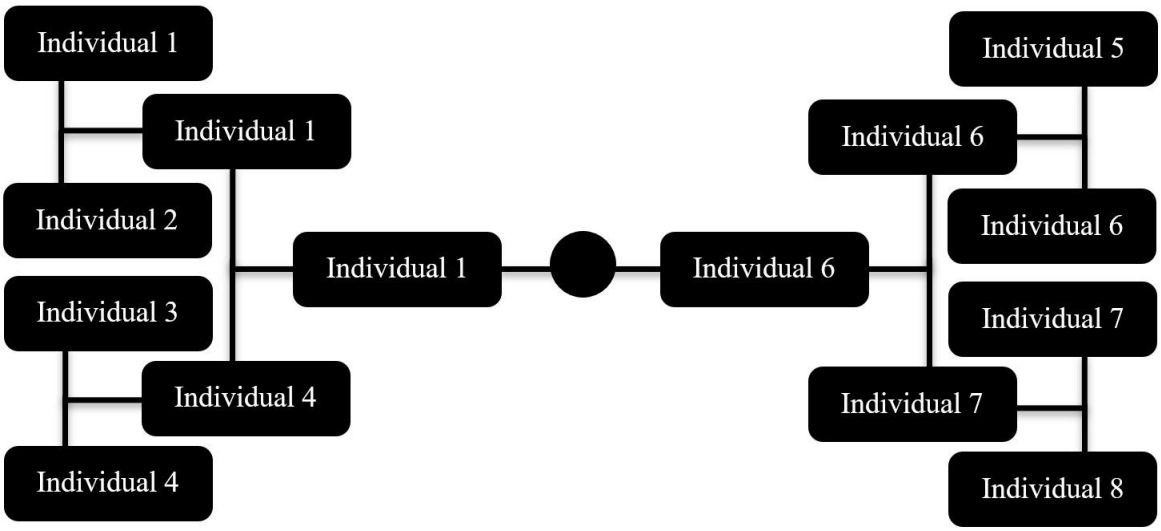

Figure 2.3. Tournament selection method (example with two individuals per tournament)

Other selection methods are rank selection [32] [35], Stochastic Universal Sampling (SUS) [34] [35], elitism [32] [33], truncation selection [35] and niching/crowding selection [33].

The crossover phase consists of merging the characteristics from two pre-selected parents in order to create a new individual for the next generation. The principal crossover methods can be divided into two basic classes depending on whether the objective is to cross individuals made of binary arrays or vectors of real numbers. Among the binary methods we have the single-point crossover and the two-point crossover [32] which segment the genotype on one or two points respectively. Also, another well-known binary method is uniform crossover which takes bits randomly from both parents.

But the operators that are of most interest for this work are those that handle vectors of real numbers (the type of individual that is present in the evolutionary strategies on which the HSA algorithm is based). One of the most classical crossover methods for this type of individual is the arithmetic crossover [36] which makes lineal combinations from the parents. The blend crossover [37] is another basic method that is simple but very effective because it is based on the arithmetic crossover, but it is also modified to work across a wider range of possibilities beyond the usual exploration range around the parents (Figure 2.4). The Simulated Binary Crossover (SBX) [37] is another well-known method that takes the idea behind the binary crossover methods and adapts it to the vectors of real numbers.

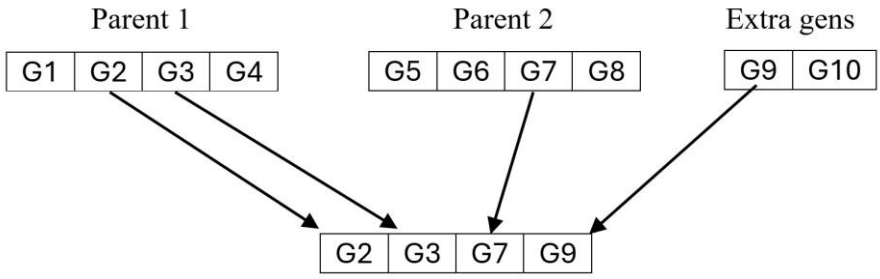

Figure 2.4. In the blend crossover, a new individual is made of a combination of gens from both parents but also new gens

Mutation is the operation where certain genes of an individual are changed randomly. Like in the crossover operation, the most well-known methods for the mutation phase can be divided depending on whether they are to be used on binary arrays or vectors of real numbers. The bit-flip and uniform mutation [32] are the simplest binary mutation methods. Uniform mutation has also a counterpart for use with vectors of real numbers which is in fact the most basic method in handling this type of individual along with the gaussian mutation [38] which mutates each gen (x_i) through a normal distribution ($\mathcal{N}$) adding a gaussian noise using a standard deviation σ (Equation 2.1).

$$x_i' = x_i + \mathcal{N}(0, \sigma^2) \tag{2.1}$$

On the copy operation there are no specific methods, as it simply consists of transferring individuals without any change from one generation to the next (Figure 2.5).

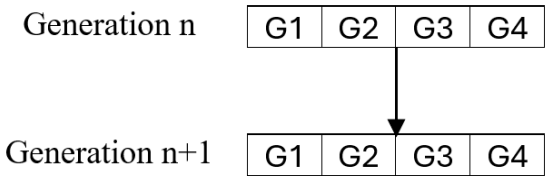

Figure 2.5. In the copy operator, an individual from previous generation is copied to the new generation without any change conserving its exact gens

An evolutionary algorithm only stops until it achieves a stop condition which is essentially the goal for which the algorithm was made. To know if the algorithm reached this goal/stop condition, it usually has a fitness function that evaluates the individuals to determine which one has the highest or lowest values and determine which are the best depending on the type of problem we want to resolve (a maximization or a minimization problem).

Within the different branches of evolutionary algorithms, those of greatest interest to this research are evolutionary strategies, which focus on individuals with continuous values and on the use of mutation over crossover. The most well-known evolutionary strategies are μ, λ and μ + λ [39]. In the μ, λ strategy, only the individuals from the offspring population (λ) survive to the next generation (Figure 2.6), while in the μ + λ strategy the algorithm chooses the best individuals from the offspring population as well as from the parent population (μ) (Figure 2.7) whose individuals can survive a k number of generations.

Figure 2.6. The μ, λ strategy

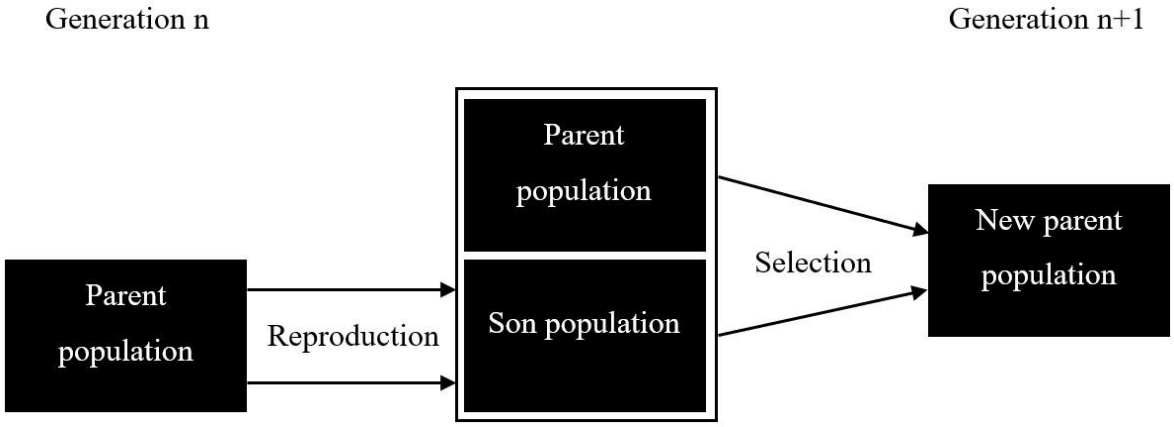

Figure 2.7. The μ + λ strategy

2.2 Swarm intelligence algorithms

Swarm intelligence algorithms are a branch of bio-inspired computing (Figure 2.8) which is based on the collective behavior of populations where the individuals exchange information with each other. Some swarm intelligence algorithms employ an evolutionary process in which during each generation, individuals in the population inherit their best characteristics to their offspring. However, in this case, individuals are influenced by the characteristics of other individuals, not only from the previous generation but also from their own. Examples of swarm intelligence algorithms are PSO (Particle Swarm Optimization) [17], ACO (Ant Colony Optimization) and BA (Bat Algorithm) [30].

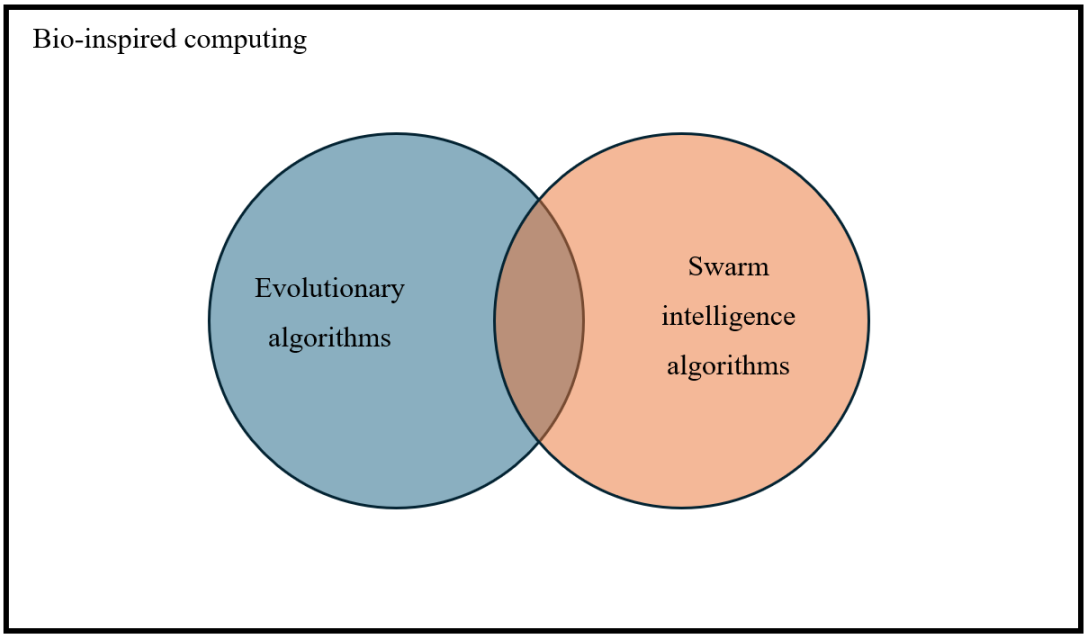

Figure 2.8. Evolutionary algorithms and swarm intelligence algorithms are both branches of the bio-inspired computing area and share common characteristics

Among swarm intelligence algorithms, the one that has stood out the most for laying the foundations of this area of bio-inspired computing is PSO. This is an algorithm that is not based on a group of animals or a specific natural phenomenon but rather takes the analogy of the general behavior of a particle swarm seeking to find an optimal solution to a given problem. The PSO algorithm works by using a set of solutions, also known as particles, which are evaluated and refined in each iteration of the algorithm until the best result or the stop condition is reached (Figure 2.9).

In PSO, each particle has four elements to work with: position, fitness, velocity and pbest. Position is the literal representation of the solution itself, fitness is the is the proximity that the position has to the truly best possible result (literally its fitness value), the range of motion that the particle has to explore other possible solutions in each iteration, and pbest is the best solution/position that the particle has had in its existence and it's replaced when a new best solution is found. Each particle starts with a random position and random velocity, and they are refined in each iteration. The best global solution for the population is called gbest and it is updated in each iteration too.

```
Initialize particles
Evaluate fitness
First pbest assignment
First gbest assignment
While not stop condition:
      For each particle:
            Update velocity
            Update position
            Evaluate fitness
            Update pbest
      Update gbest
End
```

Figure 2.9. General PSO pseudocode

Another example of a swarm evolutionary algorithm is the HSA algorithm [31] which is inspired by how bees search and collect food in order to find solutions in the most efficient conceivable way considering three phases: exploration, recruitment and harvesting (Figure 2.10).

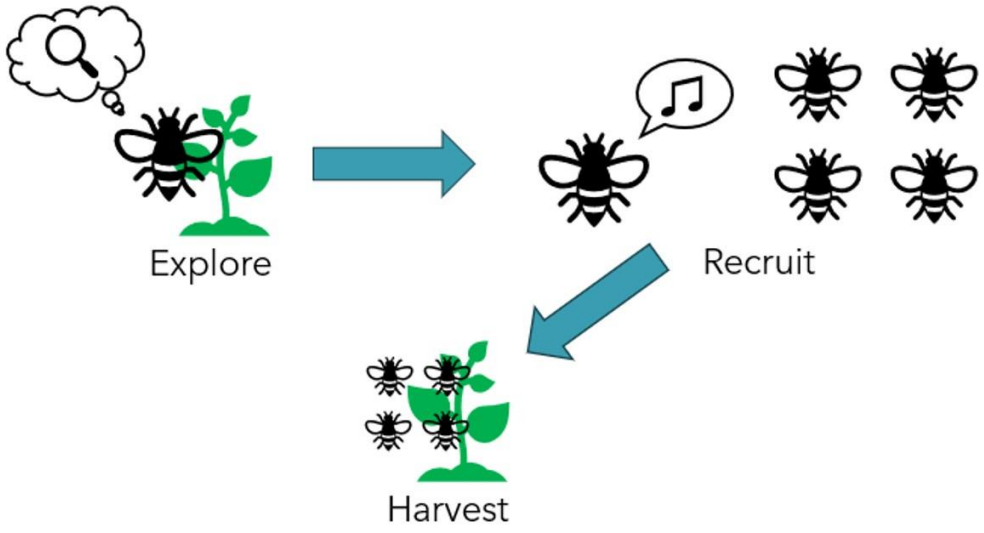

Figure 2.10. General process of how bees search and collect food

In exploration phase, a group of explorer bees travels through a search area looking for food. If an explorer bee finds a promising area, it tries to recruit more bees (starting the recruitment phase) to make the most of that food source. If the bee does not find food or fails trying to recruit new bees, then it enters a state of inactivity. If the bee found a promising food source and had success recruiting bees to exploit it, then the harvesting phase begins (Figure 2.11).

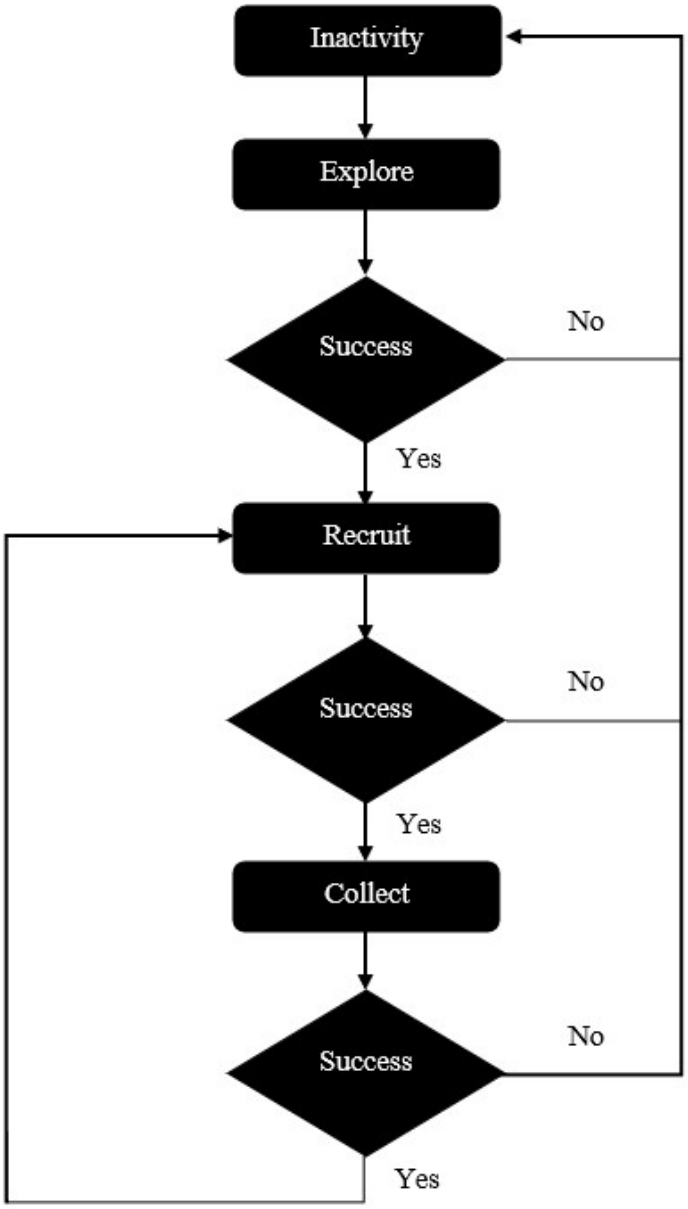

Figure 2.11. General bee life cycle inside the HSA algorithm

Besides being a swarm intelligence algorithm, HSA is an evolutionary algorithm where the previously mentioned exploration-recruitment-harvesting process is repeated generation after generation in order to find the best solution to the problem the algorithm tries to resolve. For the evolutionary process, HSA works as an evolutionary strategy too

employing the μ, λ or the μ + λ strategies to each HSA phase (exploration, recruitment and harvesting) (Figure 2.12).

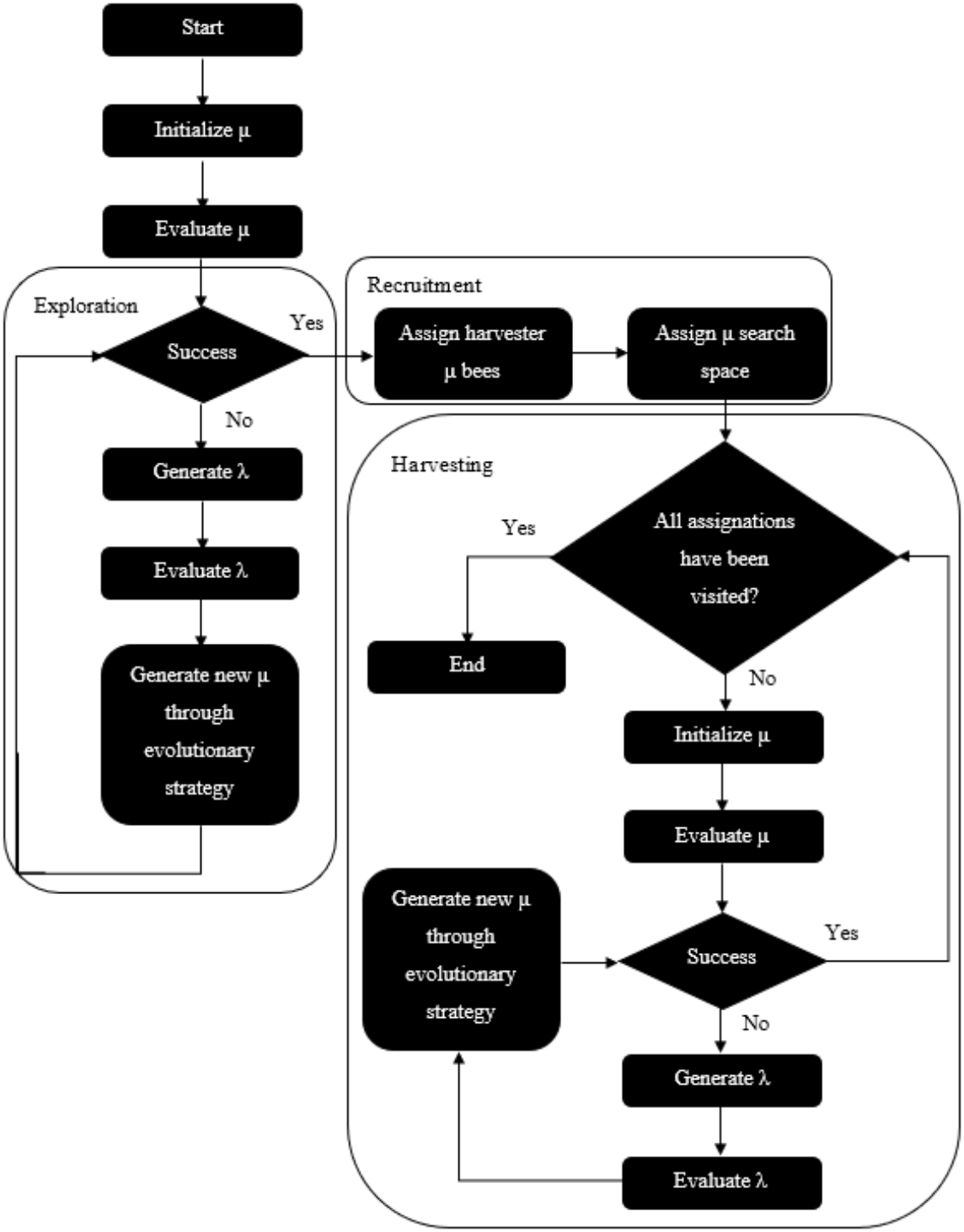

Figure 2.12. The three phases of the HSA algorithm

As was stated in the previous chapter, among the computational tasks that have been attempted to be solved using the HSA algorithm is the VOT [15] [16] [17]. The idea behind using the HSA algorithm for VOT is that the "bees" would follow the object frame by frame and in each one land on the coordinates where the target should be, just as real bees would land on a food source in real life (Figure 2.13).

Frame n Frame n+1

Figure 2.13. Graphic representation of how HSA's bees/individuals (yellow points) follow an object through a video sequence

Each bee has two genes/attributes which are x and y coordinates, along with two predetermined numbers that represent the object's weight and height. These four attributes (x, y, weight and height) conform the bounding box (with x and y being the top left corner coordinates) (Figure 2.14), a rectangle that represents the (possible) object location.

To determine if an individual has found the correct location, HSA is supported by the ZNNC algorithm which is used as fitness function. ZNCC algorithm receives a previous image (template) of the object we are looking for and apply its image correlation to compare this to the bounding box of each bee and determine how similar the bee's region is to the object's original template. Since this is a maximization problem, the bee with the highest correlation value (ZNCC's γ value) will be the one with the most likely location for the object.

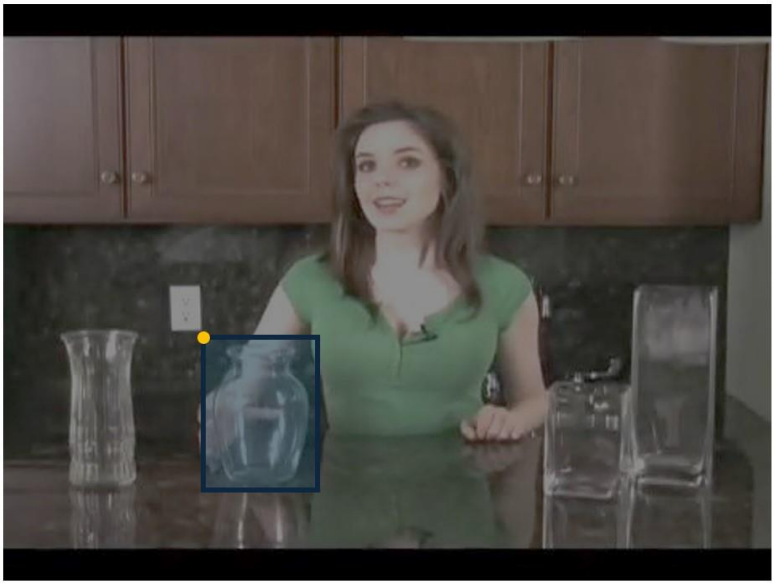

Figure 2.14. Example of a bee (yellow point) and the bounding box that it represents (blue box)

In the traditional HSA implementation for VOT [15], the complete exploration-recruitment-harvesting cycle is done every frame with all the generations of the evolutionary process inside exploration and harvesting phases (Figure 2.15). First, the explorer bees try to find the object on the frame, the explorer bees evolve through generations until it is time to pass to recruitment phase where harvester bees are assigned to the best explorer bees' locations. These harvester bees try to find the fittest location in the harvesting phase having their own evolutionary process.

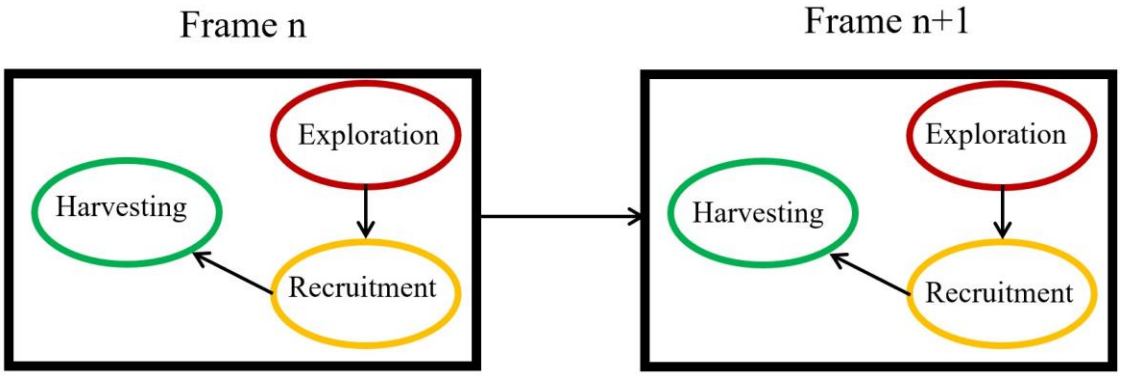

Figure 2.15. The three HSA phases (exploration, harvesting and recruitment) are done in each frame

In the new implementation of HSA proposed in this work (and which will be detailed in the following chapters), the goal is to optimize the algorithm so that it is no longer necessary to carry out the three traditional phases of the HSA algorithm in each frame, thus substantially reducing the execution time and increasing the probability of finding the target in each frame by reducing noise and making object searches in a smartest way.

Chapter 3

Approaches and Solutions of the Video Tracking Problem

Over the years, different solutions have been proposed to solve the VOT problem by addressing different approaches, from the most traditional and basic ones that arose with the technical limitations of the early days of computing, artificial intelligence and computer vision, to the most complex and sophisticated ones that use the latest software and hardware tools.

3.1 Video tracking systems classification

Currently, there is no consensus within the computing community that establishes a definitive standard classification for VOT methods. Different works attempt to establish their own classifications based on the specific needs of their particular case studies [11] [40] [41].

Considering the important role the ALOV300++ dataset plays in the current investigation; we will focus on the VOT systems review made by Smeulders et al. in the previously mentioned paper “Visual Tracking: An Experimental Survey” (2014) [11] where ALOV300++ dataset was proposed for the first time. In this work, the authors divide VOT systems into two main categories: online and offline. The online VOT systems are those that deliver processing results for each frame during runtime, while offline systems are

those that deliver the overall results of processing the entire video until the end. Offline systems are often used only for experimental purposes because of their impractical nature for real world applications, while online systems are suitable for use in everyday practical applications like robotics, mobile commercial devices and IoT (Internet of Things) devices, even if not all the online video trackers deliver results in real time.

Due to the nature of this research and the method proposed here, we will focus on the online VOT algorithms and leave aside offline algorithms, which include some of the most popular recent methods such as some deep learning algorithms (which can fall into both the online and offline categories in their VOT implementations). Having said that, Smeulders et al. propose a five-category classification system for the online trackers:

1. **Tracking using Matching:** this is the most basic category where the algorithm looks for the object in the current frame based on the previous frame(s). E.g., Normalized Cross-Correlation (NCC) [42], Lucas-Kanade Tracker (KLT) [43], Kalman Appearance Tracker (KAT) [44], Ocean [56].
2. **Tracking using Matching with Extended Appearance Model:** in this category, video trackers try to maintain an extended model of the object and/or its behavior in the previous frames and its possible variations. E.g., Incremental Visual Tracking (IVT) [45], Tracking on the Affine Group (TAG) [46], Tracking by Sampling Trackers (TST) [47], UpdateNet [57].
3. **Tracking using Matching with Constraints:** trackers that belong to this category try to use a sparse, simple and less complex representation of the object in order to optimize the algorithm's performance. E.g., Tracking by Monte Carlo sampling (TMC) [48], Adaptive Coupled-layer Tracking (ACT) [49], L1-minimization Tracker (L1T) [50], STARK [58].
4. **Tracking using Discriminative Classification:** this type of trackers attempts to distinguish object's pixels against background's to create an object's model that is updated frame by frame. E.g., Foreground-Background Tracker (FBT) [51], Hough-Based Tracking (HBT) [52], Super Pixel tracking (SPT) [53], DiMP [59].

5. **Tracking using Discriminative Classification with Constraints:** this category of methods is like the previous one but focusing their efforts on creating a correct classification of the pixels of the object(s) instead of focusing their attention on its location. E.g., Struck [54], Accurate Tracking by Overlap Maximization (ATOM) [60].

ZNCC (Zero Mean Normalized Cross-Correlation) is by itself a tracking method that belongs to the first category (trackers that use matching) like its predecessor on which it is based (NCC). This is a method created to measure the statistical similarity between two matrices/images (Figure 3.1).

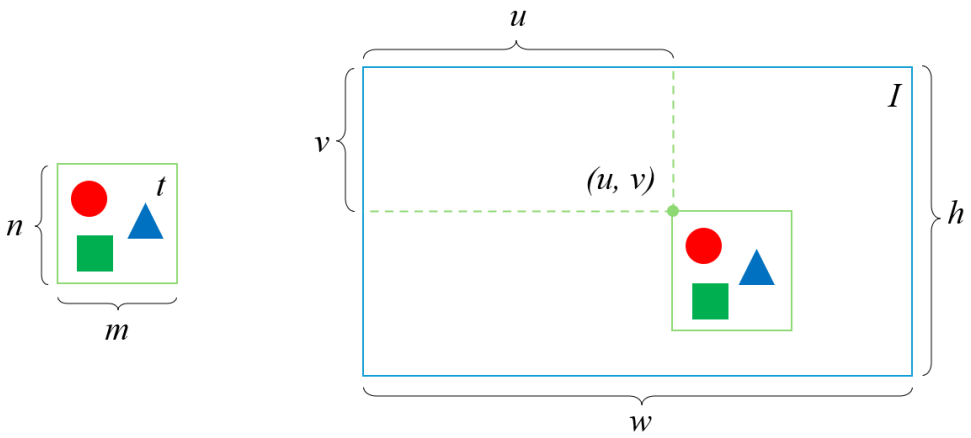

Figure 3.1. Image matrix template defined by (n x m) is compared to the section at (u, v) on the image/frame I defined by (w x h)

How ZNCC gets the similarity between the two matrices is defined by the next equation (Equation 3.1):

$$\gamma(u,v) = \frac{\sum_{x,y}\left[I(x,y) - \bar{I}_{u,v}\right]\left[t(x-u,y-v) - \bar{t}\right]}{\sqrt{\sum_{x,y}\left[I(x,y) - \bar{I}_{u,v}\right]^2 \sum_{x,y}\left[t(x-u,y-v)\right]^2}} \tag{3.1}$$

where γ is the final similarity value between the comparison template and the analysis area defined by (u, v) which are the coordinates of the area's top left corner. *I(x, y)* refer to all the pixels that compose the analysis area, $\bar{t}$ (Equation 3.2) is the average template value and $\bar{I}$ (Equation 3.3) is the value obtained through the sum of pixel values in an area and divided into the size of that particular area (m x n).

$$\bar{t} = \frac{\sum_{x,y} t(x-u, y-v)}{m \times n} \tag{3.2}$$

$$\bar{I}_{u,v} = \frac{\sum_{x,y} I(x,y)}{m \times n} \tag{3.3}$$

Using this statistical equation, ZNCC measures similarity between an image of the object a.k.a template, and different regions of the actual frame in order to find the object. This template is usually extracted from the previous frame by cropping the region where the object was found in that frame.

On its primary traditional implementation to VOT, HSA (Honeybee Search Algorithm) [15] corresponds to the first category of the online VOT classification proposed by Smeulders et al. since its mechanism to recognize the target object in a frame (a.k.a fitness function in the context of evolutionary algorithms) is ZNCC, something that makes HSA a matching VOT algorithm that in this case is powered up by the evolutionary swarm characteristics of the HSA metaheuristic.

Due to the nature of the improvements that will be detailed in following chapters, the version of the HSA algorithm for VOT proposed in this research may instead fit into the second category since it tries to keep most extended model of the object and its behavior frame by frame. It is worth mentioning that even if this paradigm followed by NCC, ZNCC, and eventually the HSA-ZNCC implementation, represents the object through a single bounding box, it is not the only object representation approach that can be found. There are many ways an object can be computationally represented but the three basics are: single point, bounding box and target silhouette [11] [19]. These three forms can also be used multiple times in the same frame to track more than one object or track multiple parts of the same object (Figure 3.2).

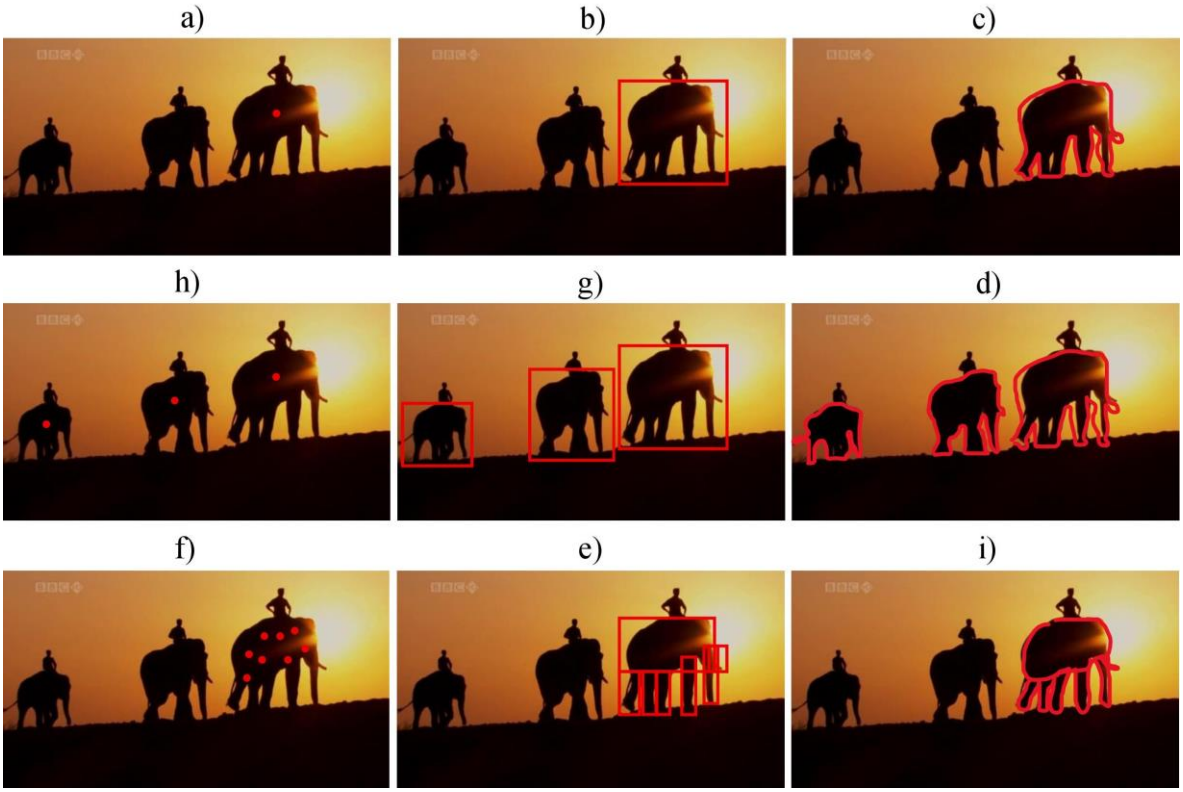

Figure 3.2. Different forms to represent an object in VOT: a) single point, b) bounding box, c) target silhouette, d) multiple points on multiple targets, e) multiple bounding boxes on multiple targets, f) multiple silhouettes on multiple targets, g) multiple points on one single object, h) multiple bounding boxes on multiple parts of one single object, i) multiple silhouettes on multiple parts of one single object

3.2 GPU acceleration in video tracking

Advances in electronics have allowed us to create new and more complex and sophisticated hardware components beyond the traditional CPUs. Components like FPGA (Field Programmable Gate Array), SoC-FPGA (System-on-Chip FPGA), MPSoC (Multi-Processor System-on-Chip) and ASIC (Application-Specific Integrated Circuit) have been used in order to enhance VOT algorithms. One of the specialized hardware components that has especial interest is the GPU (Graphic Processing Units), an increasingly common component in everyday computers and devices and specifically designed to process graphic elements especially tridimensional graphics but also images, videos and even non-graphic elements such as cryptocurrency mining operations.

To take advantage of GPUs, VOT algorithms are re-designed to work through what we know as parallel computing, a computational approach that takes advantage of the multiprocessors' nature present in modern computers dividing a complex process in multiple single and more simple processes called threads and that can compute in parallel a lot of instances of the code we want to parallelize and which is called kernel (Figure 3.3).

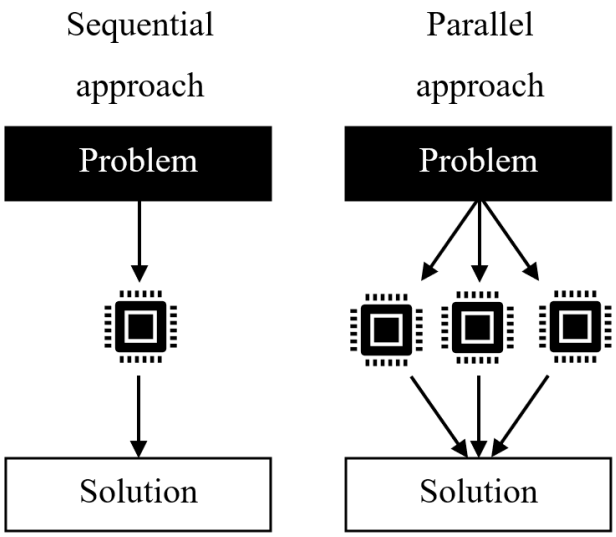

Figure 3.3. The traditional sequential approach uses only one processing core to achieve a task while parallel approach uses multiple cores

Michael J. Flynn proposed the Flynn's taxonomy in 1972 [55], a four-category classification for parallel computing techniques based on how parallel computational systems distribute tasks across computational processing units:

1. **Single instruction, single data (SISD):** this category refers to the traditional non-parallel sequential approach where we just have one instruction that works over one single data piece.
2. **Single instruction, multiple data (SIMD):** this category refers to how components execute a single processing instruction over multiple amounts of data. This is especially used when we have a lot of data to process (like databases or big matrix operations in graphics processing).
3. **Multiple instructions, single data (MISD):** here different operations are executed in parallel over the same input data. Especially used in signal processing.

4. **Multiple instructions, multiple data (MIMD):** this is the maximum parallel computing expression where multiple instructions are performed on different amounts of data. Especially used in clusters.

Depending on its architecture, a GPU usually executes tasks according to the second category (SIMD). A GPU is essentially an array of interconnected processors a.k.a vector units that share information between each other through global memory which can be accessed by any processor, and local memory which can only be accessed by certain specific processors that are allowed to, this because the GPU processors are divided into different groups called work blocks. Usually, GPU operations are coordinated by the CPU which remains as the principal processing component of the computer/device. The growing popularity of GPUs has led to the emergence of various libraries, frameworks, APIs, and other software platforms and tools that facilitate the programming of algorithms designed to run on GPUs. Some popular current platforms are CUDA [79] and OpenCL [80], which are also designed to fully exploit the parallelization potential of GPUs.

In video tracking, parallelization can be used to improve algorithms' performance in several ways depending on the nature of the algorithm or method itself. In the case of matching algorithms, and specifically ZNCC on its pure exhaustive form, we can leverage parallelization to analyze the correlation between the template and different frame areas simultaneously (Figure 3.4).

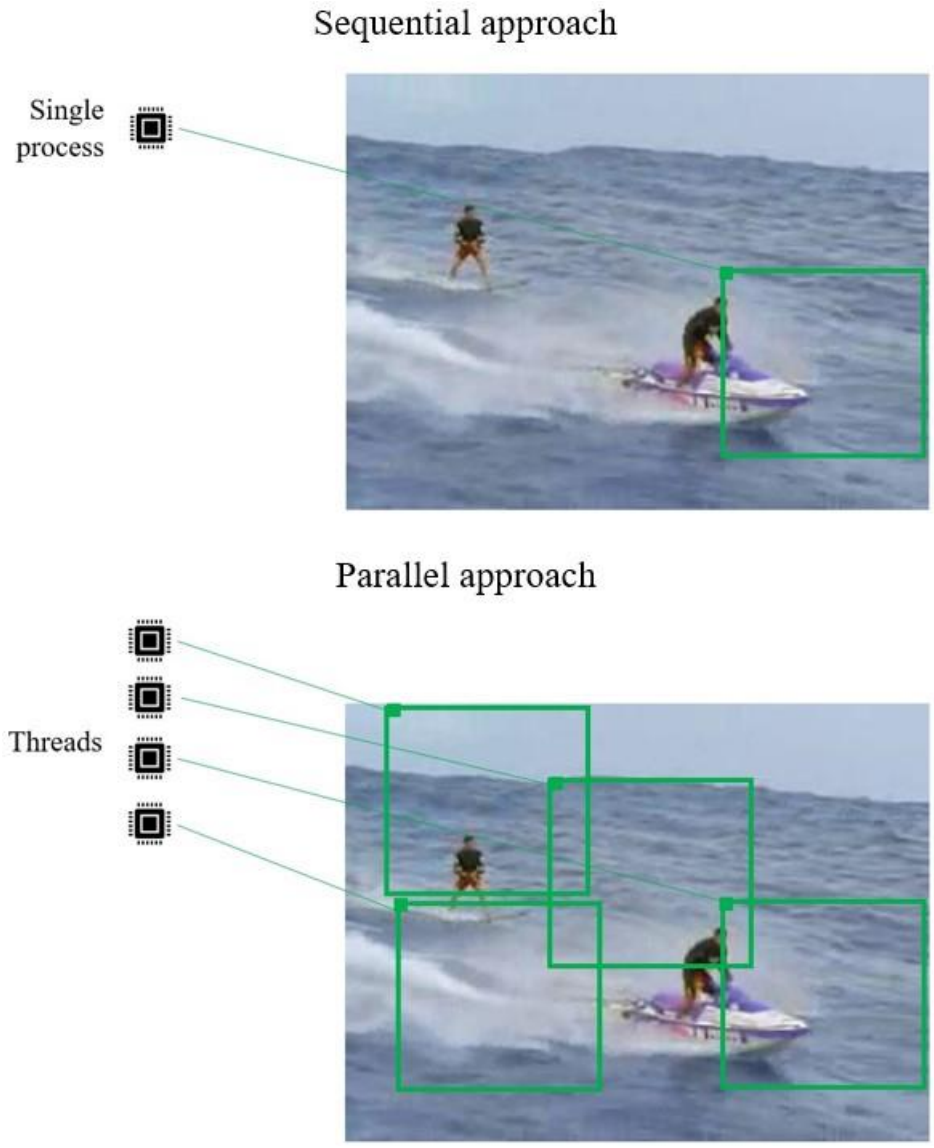

Figure 3.4. ZNCC executed through a sequential approach would require analyzing one frame zone at a time, while ZNCC executed through a parallel approach allow us to analyze different zones simultaneously

3.3 Parallel swarm and evolutionary algorithms

Parallel processing is an approach that can be used on swarm intelligence and evolutionary algorithms too. Also, as mentioned earlier, the use of GPUs in different branches of computational R&D has become increasingly common, and swarm intelligence (SI) and evolutionary algorithms are no exception [61] [62] [63] [65].

In their 2016 paper titled "A Survey on GPU-Based Implementation of Swarm Intelligence Algorithms" [64], Ying Tan and Ke Ding propose a four-category classification system for parallel SI algorithms based on GPU (Figure 3.5):

1. **Naive model:** this is the simplest model where we only parallelize the fitness which is usually the phase that consumes more time and resources, and we don't care about its internal working; we just take it as a black box ready to be parallelized in both a coarse-grained (task-parallel) or a fine-grained (data-parallel) approach [68] depending on the situation of the problem and the data magnitude. There are several examples of this model especially with the PSO algorithm.
2. **Multiphase model:** as its name suggests, in this approach we must parallelize more operations than just the fitness function like we do in the previous category, but some operations remain executed under a sequential approach on the CPU. This model allows parallelization to be adapted to algorithm's tasks and create different custom kernels with the appropriate granularity. Unlike the naive model, multiphase model is not compatible with all SI algorithms.
3. **All-GPU model:** in the previous category the SI algorithm is partially parallelized on the GPU while this third category seeks to parallelize the entire SI algorithm, executing all its operations on the GPU (including serial code that cannot be parallelized) and leaving the CPU with the sole function of delivering the algorithm's parameters and receiving its results. This resolve latency problems that could appear in CPU-GPU communication but also could create new synchronization problems in big swarm algorithms that make the algorithm encapsule a big number of operations (or even all of them) into a single processing block since synchronization between blocks in GPU is not available.

4. **Multiswarm model:** as previously stated, the all-GPU model is a good option for SI algorithms with a relatively reduced swarm population, but if the algorithm works with large populations, this previous model could suffer problems due to the hardware synchronization which is not available on the entire GPU but can only be carried out by processes that share the same work block. Tan and Ding [64] propose two solutions to this issue which are remove data-dependency and use software synchronization but also propose this fourth final GPU parallel model: the multiswarm model, which executes different instances of the same SI algorithm on different blocks which deliver their individual results to CPU who has to select the best result(s).

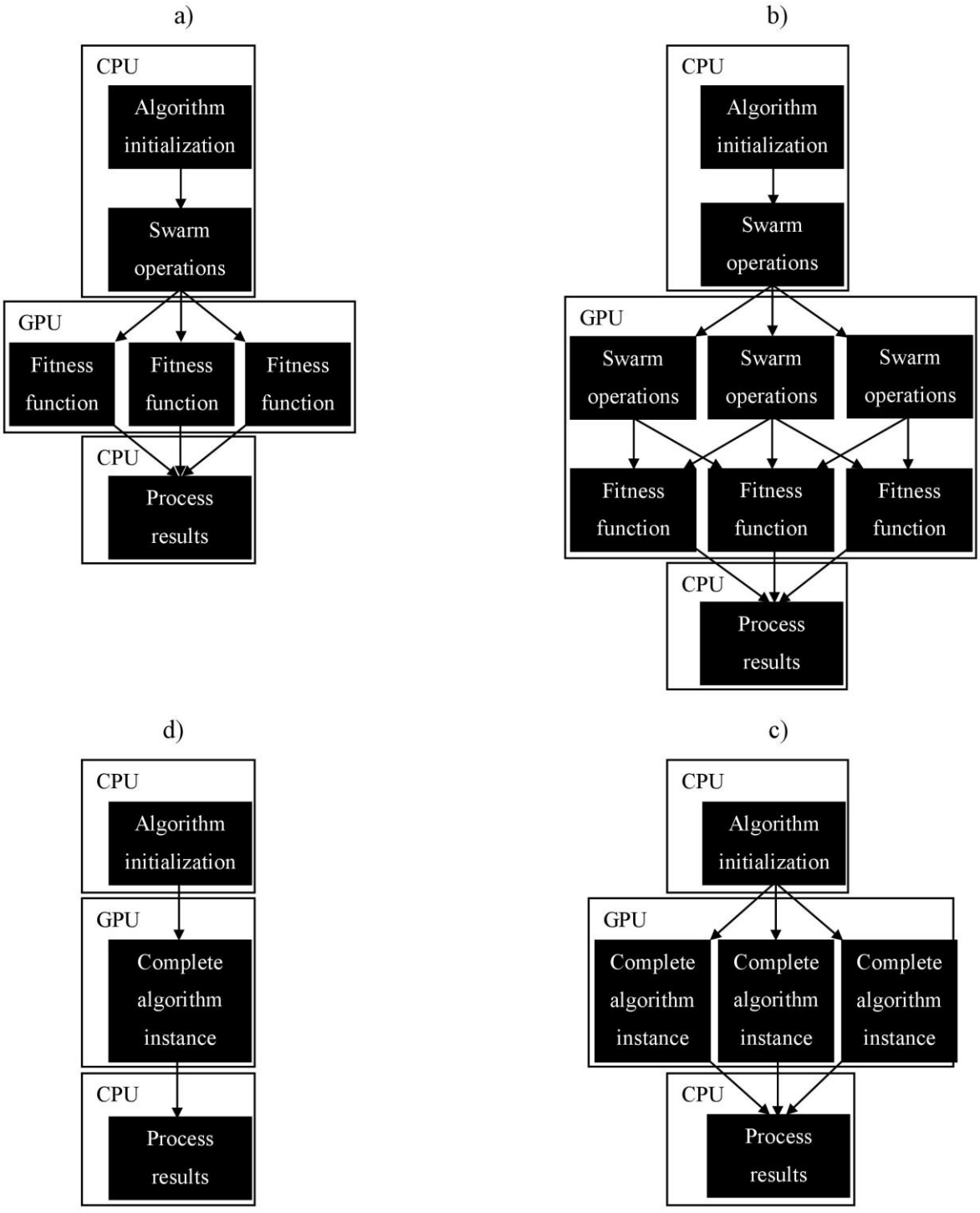

Figure 3.5. a) Naive model, b) Multiphase model, c) All-GPU model, d) Multiswarm model

The original HSA implementation for VOT [15] took an all-GPU parallel model but as it will be described in the following chapters, the current implementation that is proposed in this implementation may took a multiphase model since the present investigation showed that some specific phases of the HSA algorithm could be executed more optimally if they were executed in this way.

On the side of the evolutionary algorithms (EA), Erick Cantú-Paz proposed in 2001 a three-category classification for parallel evolutionary algorithms (Figure 3.6) on his book "Efficient and Accurate Parallel Genetic Algorithms" [66]. Even if this taxonomy was established thinking about parallelization on CPU only (since Cantú's book was published in the early 2000s), it is compatible with parallelization on GPU [67]:

1. **Master Slave Model:** like in sequential EA implementations, this EA parallel model is based on the use of one single population but managed by a master process. Every individual of the population (called "slave process") is executed in parallel and deliver its results to the master process. Usually, the fitness function is the phase that is computed in parallel into the slave process, but (unlike the naive model of the parallel SI algorithms) the master slave model is not limited to that and allows genetic operators to be implemented in parallel too.
2. **Island Model:** this model looks for work with different sub-populations a.k.a islands, instead of just one large population. Each population works independently and all of them are interconnected through certain "emissary" individuals that "migrate" from one population to another sharing information. This model is usually used on networked clusters.
3. **Cellular Model:** in this model all the population is arranged in a one-dimensional or two-dimensional grid where the individuals are still processed in parallel but can only share information with their immediate neighbors. Usually used on computer networks based on SIMD and MIMD parallel processing models.

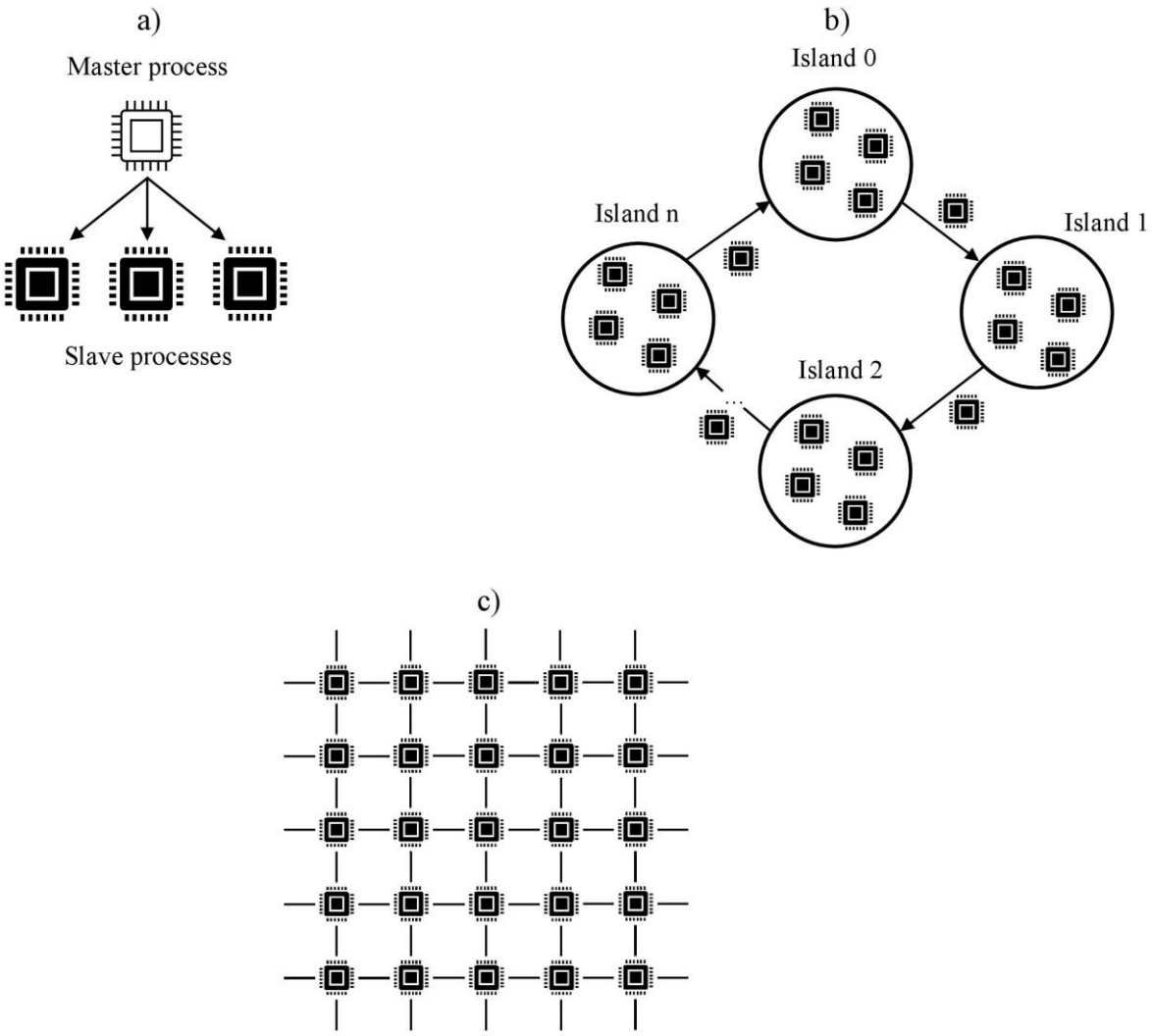

Figure 3.6. a) Master slave model, b) Island model, c) Cellular model

Original HSA implementation for VOT [15] was designed under a basic master slave model, something that is maintained in the implementation proposed in this thesis as it will be seen in the following chapters.

3.4 Video tracking evaluation

Different methods have emerged to evaluate VOT systems. A VOT system is usually evaluated based on its ability to correctly locate an object within a video frame, and speed is literally how fast the algorithm is when it tries to locate the object(s).

Regarding the capability of the system in performing the VOT task, we have two types of metrics depending on the system's ability to locate the object in a single frame, and its ability to track it throughout the entire video. The first one rates the success of the algorithm locating the object in one single frame while the second one rates the success of the algorithm on all the frames and how successful it was tacking the object thorough the entire video; this second type usually employs one frame metric to know how much success the system had on the whole video. Some of these single frame metrics are:

- Intersection over Union (IoU): percentage in which the bounding box of the calculated location (truth) and that of the actual location (ground truth) coincide.
- GIoU: this is a variation of the traditional IoU that is modified in order to identify new characteristics such as the distance between bounding boxes and the size change between them.
- Center Location Error (CLE): Euclidean distance between the center of the truth's bounding box and the center of the ground truth's.

By the other hand, some general whole video metrics are:

- Precision: percentage of true positive predictions against false positives.
- Recall: percentage of true positives against false negatives.
- F-Score: harmonic mean between precision and recall
- Success rate: percentage of frames where the prediction reached a personalized IoU threshold.
- Failure rate: number of times the algorithm lost the object.
- Fragmentation: number of times the tracking was interrupted.
- Equivalent Filter Operations (EFO): rates the computational cost of a tracker based on its relative algorithmic complexity regardless of the hardware

To rate speed, the most common metrics are:

- Frames per second (FPS): how many frames the algorithm manages to process in one second.
- Average time per frame: the time it usually takes the algorithm to process a frame.

It is worth mentioning that, even if all investigations look to get the highest possible score in both metrics, their objectives usually depends on the applications for which it is considered, the state of the art and precedents that are considered, and the evaluation benchmark that is used but it is also worth mentioning that nowadays there are not a single perfect tracker 100% successful. Some of the results obtained on the ALOV300++ benchmark are described in Chapter 1 where we take a look at the initial tests made by Smeulders et al. in 2014 [11] and where Struck were the best tracker obtaining a 66% F-Score, but more recent works report a higher score on this benchmark [2] [71] [72]. On other considerably recent benchmarks, high scores have been observed as is the case with SoTA [73], which obtained an >80% success in the GOT-10k benchmark [74].

On the speed goals side, even if achieve real time (image processing speed of human eyes) is the goal that all video trackers would like to achieve, it is worth to mention that in literature it has been found that there is not an explicit definition of speed in FPS to achieve "real time" [76] [77] [78] since it depends on the specific application to which the research is directed or if it is directed to a specific type of content such as videos or movies whose average speed ranges from 24 FPS to 30 FPS, or computer animations (mainly videogames) that can range from 30 FPS to 60 FPS or even much more. In the case of video tracking, the speed of 15 FPS is usually an acceptable measure for a video tracking system to function properly with a good time of response considering the processing times and computational obstacles it faces [75].

The original HSA implementation [15] used both F-Score and FPS to be evaluated, supporting F-Score (and consequently precision and recall too) with the use of IoU, an approach that is followed in this research as will be explained in the following chapter,

However, for more information on the metrics described above, you can refer to [8] [69] [70].

Chapter 4

Development of the Parallel Video Tracking System

The previous chapters explained the theoretical basis behind this research. This chapter will explain the development of the final VOT software, detailing the improvements made compared to the previous implementation.

4.1 Hardware and software tools selection

Having said that, to accelerate the entire HSA - ZNCC system, we seek to take advantage of GPUs and their specific components designed for the graphic resources processing that can also be used to improve parallel programming in HSA. To achieve this, the computer(s) where the system would be programmed and tested must have the following characteristics:

- CPU: x64 architecture capable of executing modern Python and C distributions including libraries to process images, carry out evolutionary algorithms and achieve parallel processing.
- Hard drive: 64 GB or higher.
- RAM: 8 GB or higher.
- GPU: 4 GB or higher, compatible with Python and C parallel computing libraries.

That said, in the following table (Table 4.1) the characteristics of the computer on which the system was programmed are compared with the one on which it was tested, and

with the computer on which the previous implementation of the system [15] was tested (in the context of this research) and with an example of a modern computer with some of the most sophisticated characteristics:

	PC where the previous version was tested	PC where the current version was programmed	PC where the current version was tested	Modern PC
CPU	Quad Core Intel Xeon	AMD Ryzen 5	Intel Core i5	AMD Ryzen 9
GPU	NVIDIA Quadro K600 (1 GB)	NVIDIA GeForce GTX 1650 (4 GB)	NVIDIA GeForce GTX 4060 (8 GB)	NVIDIA GeForce GTX 5090 (32 GB)
RAM	8 GB	16 GB	32 GB	64 GB
Hard drive	512 GB	1 TB	2 TB	4 TB
OS	Ubuntu 14.04 LTS	Windows 10	Windows 10	Windows 11

Table 4.1. Computer comparison

Even if the system was programmed and developed in computers with NVIDIA GPUs and Intel and AMD CPUs, it is not designed to work exclusively with these CPU and GPU models because the aim is for the code to be as open and portable as possible The goal is for the code to be as open and portable as possible, without being limited to a specific component or computer brand or model.

Program code was developed mainly in Python. While Python has the disadvantage of being an interpreted language (which could affect processing speed), it has also proven to be one of the best languages for artificial intelligence development and data analysis. Also, Python is a high-level language that has specialized memory and process management mechanisms (that low-level languages like C don’t have) in addition to its wide variety of optimized libraries and a large community that is constantly working on the language in order to improve it and find new ways to expand its usage.

The main library being used is OpenCL, one of the best-known libraries for implementing parallel computing in most programming languages. In the case of the Python's version, OpenCL requires the use of kernel code in C language resulting in a hybrid code whose base (core code) is in Python but implements C code in the tracking component, specifically in the ZNCC algorithm part which is believed to be the most demanding section of this implementation since it's HSA's fitness function. OpenCL was chosen over CUDA (currently the most popular GPU parallel computing library) for two main reasons: first, it was the library used in the previous implementation of the system [15], making it easy to port to Python without having to rewrite the code from scratch. Second, as mentioned earlier, the intention with the system is to be portable for use on any GPU model and using CUDA (designed exclusively for NVIDIA GPUs) or another library with brand or model restrictions would have largely eliminated this portability, something that doesn't happen in OpenCL. Although, the implementation of compiled C code into the system somewhat compensates for Python's disadvantages as an interpreted language.

In addition to OpenCL, other pure Python libraries are being used such as Joblib [88], a library specialized in the implementation of parallel computing in pure Python and which is used to parallelize some evolutionary operations in Python and that are implemented using DEAP, a specialized and optimized library for implementing evolutionary computing in Python and that has a wide range of functions that implement evolutionary strategies, genetic mutation, crossover and selection operators among other evolutionary computing tools. Another library that is being used is OpenCV, the main Python (and possibly any programming language) library designed for specialized image processing capable of reading and manipulating image data in all its formats. In addition to the libraries mentioned, the program uses some well-known libraries commonly used in Python, such as NumPy.

4.2 Component-based design

The development of this latest version of the HSA - ZNCC implementation was made taking advantage of the multi-paradigm property of Python language as well as the optimization of its specialized libraries to create a flexible and easily manageable and usable system.

The system is designed to focus on a core autonomous components that would only be responsible for executing the video sequences frame by frame in addition to coordinating the work of the rest of the components that would oversee more specific tasks such as the tracking component. which would oversee tracking the object itself and where the ZNCC code would be implemented. Similar components or components that are designated to achieve the same function will be organized into packages which in addition to keeping the code organized, will allow an easy export and integration of those components inside and outside the system.

As can be seen in the system diagram (Figure 4.1), the different components are grouped according to the functions they perform. The most important package is the trackers package because here lies the VOT algorithm. As can be seen, both the HSA and the ZNCC components are here, this is because the ZNCC algorithm can be a tracker by itself but is also used by the HSA component as its fitness function. Object representation package has now just one component: the bounding box component, which is used to represent the target object as a bounding box in the video but in future work, new components could be added to represent the object in other ways (for more information about different ways of representing an object in VOT, see Chapter 3). The video component is the one that manages the video's attributes and representation, while the ANN files component is in charge of reading .ann files, and it's used to read the files that describe the location of the targets in the ALOV300++ dataset videos which are in .ann format (more information about the ALOV dataset will be detailed later). Right now, these two components are not part of a package since they don't need to because they do unique functions that can be concentrated only in those components, there are no components that do the same and consequently there is no need to create whole packages for it.

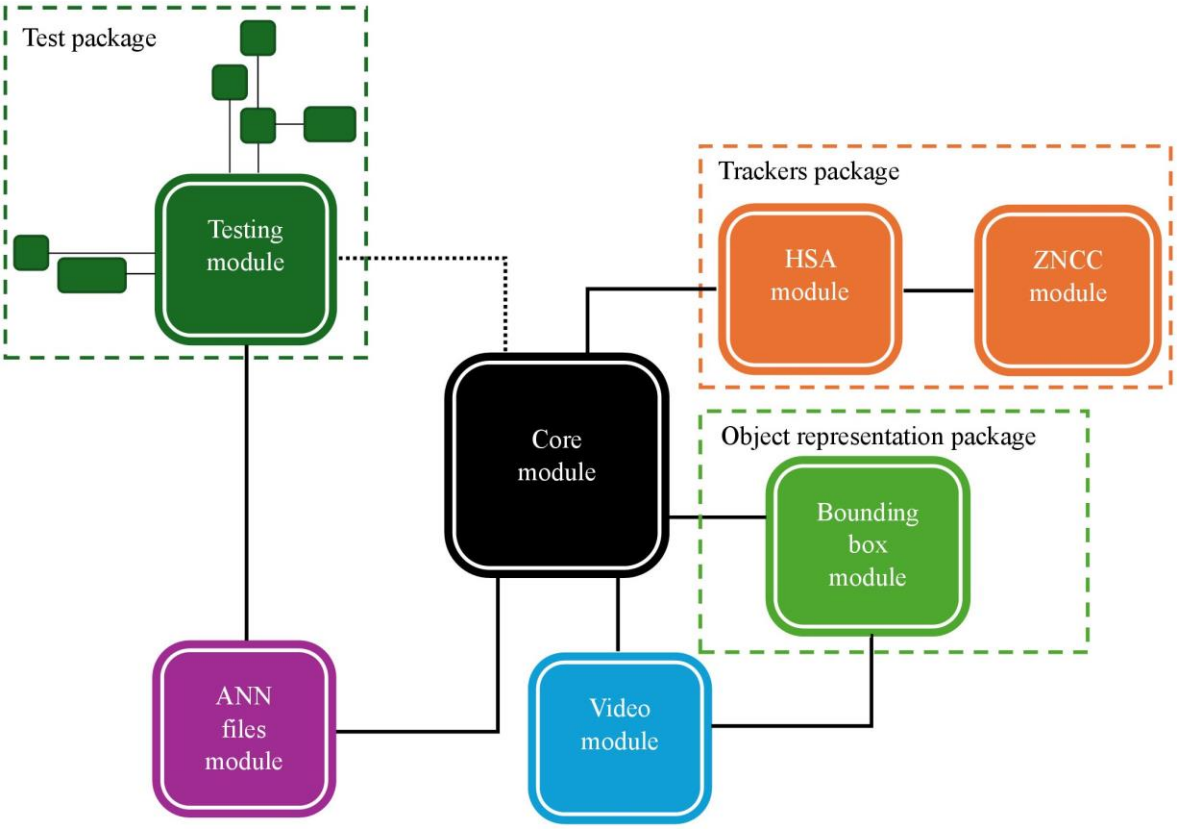

Figure 4.1. System design diagram. As can be seen, the test package contains little subcomponents which are connected to the testing component (and are going to be explained). Testing components are also connected to the core component through a dotted line representing how it can work independently (as will be explained later)

Besides the core component (and as can be seen in the general system component diagram), the only other independent component would be the testing component, as it is intended to be a system capable of running and calculating different metrics without the need to rely on the runtime analysis of the main component. Instead, this component can receive results files (in .csv format) and analyze them after the execution of the main video tracking system to subsequently calculate metrics such as F-Score. Still, this test component could be integrated into the core component to calculate the necessary metrics at runtime.

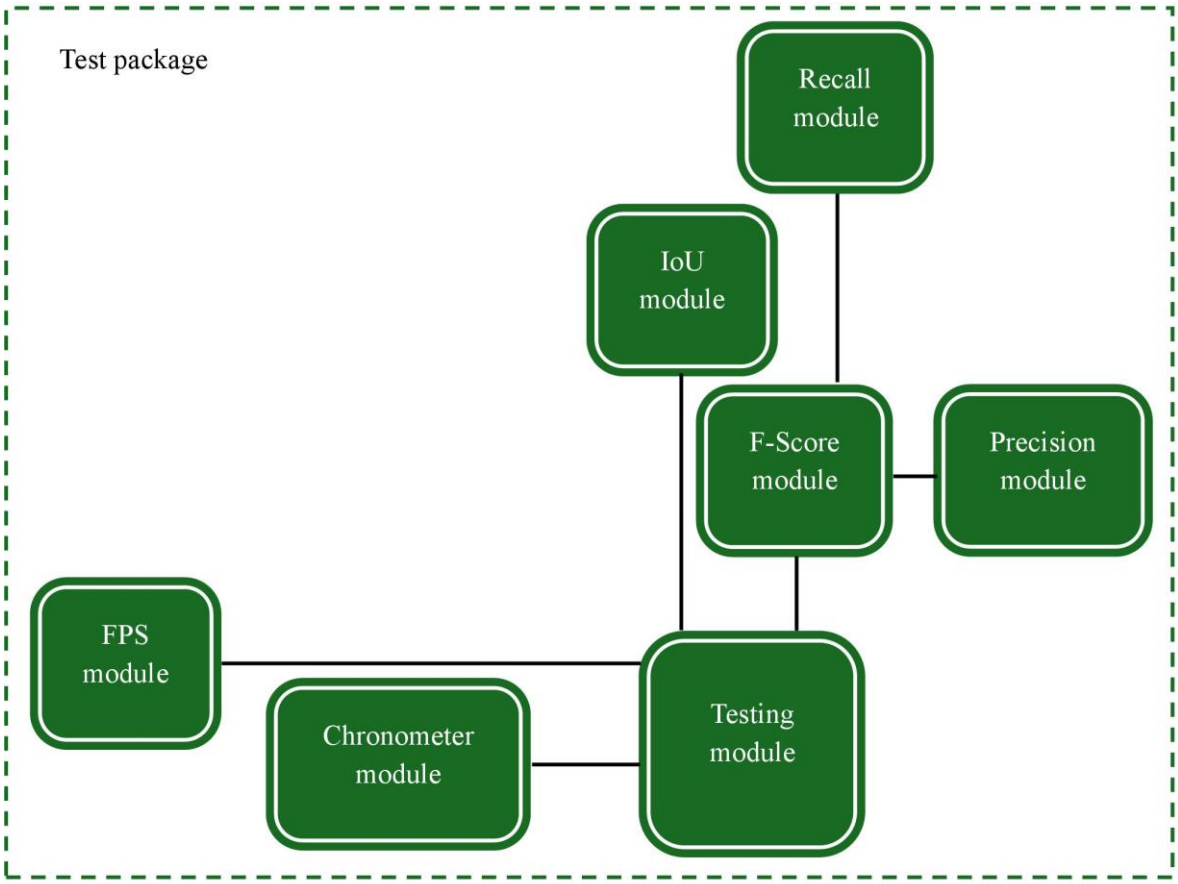

Figure 4.2. System design diagram for the testing subsystem

Due to the independence of the testing component, it can be considered a complete subsystem (Figure 4.2) with its own core component which is the testing component itself connected to the different components in charge of calculating different testing metrics such as the FPS component. As it can be seen, there is a component simply called "chronometer" which (as its name suggest) it's simply a chronometer that is used to determine time metrics such as the total system execution time or the partial execution time it took the system to process a frame and return it in different units of time that can range from milliseconds to weeks. The F-Score component is the most complex one since it depends on two other metric component which are precision and recall (as it was briefly explained in chapter 3.4 and will be explained in detail in chapter 4.5.2), it also depends on the IoU (Intersection over Union, which was previously explained in chapter 3.4 and will

be explained in detail in chapter 4.5.2 too) component, but not strictly since F-Score (or more specifically, Precision and Recall) only depends on the number of true positives (TP), false positives (FP) and false negatives (FN) which are calculated on the main testing component based on the IoU component results, so in a future work someone can replace the IoU component with another one that uses another method capable of provide the necessary information to calculate TP, FP and FN. It is worth mentioning that the F-Score component can already automatically calculate F1-Score, the most known variant of the F-Score and which is used in this research (as will be explained in more detail later). Also, the testing component is connected to the ANN files component because it requires reading the ground truth form the ANN files in order to calculate IoU.

As previously said, we took advantage of the multi-paradigm nature of the Python language. Even if at the beginning of development, the idea was to apply the oriented object programming (OOP) paradigm at all the components, later we found that the OOP paradigm was the best option for some components (especially the big ones) but in some of them the most practical option was to conserve structured programming.

The class diagram seen below (Figure 4.3) shows how the different classes of the system relate between each other. As can be seen, every class is somehow related (directly or indirectly) to the Main class which is the class that represents the main component. The HSA, Video and Reader (from ANN files component) classes are essential to the Main class correct working, even if their components can be decoupled, another component must occupy their place. E.g., the HSA component performs the role of the tracker component, that's why it's necessary, but it can be also decoupled and eventually replaced by another component able to perform that role. The same happens between the Video class and the boundingBox class which is necessary to the Video class because it needs a way to represent the target but not necessarily a bounding box, so eventually someone can replace the Bounding Box component with another one that contains another way to represent a target. Also, the boundingBox class has a dependency relationship with the Main class since this one only uses it to create a bounding box which is going to be the *target* attribute of the Video class.

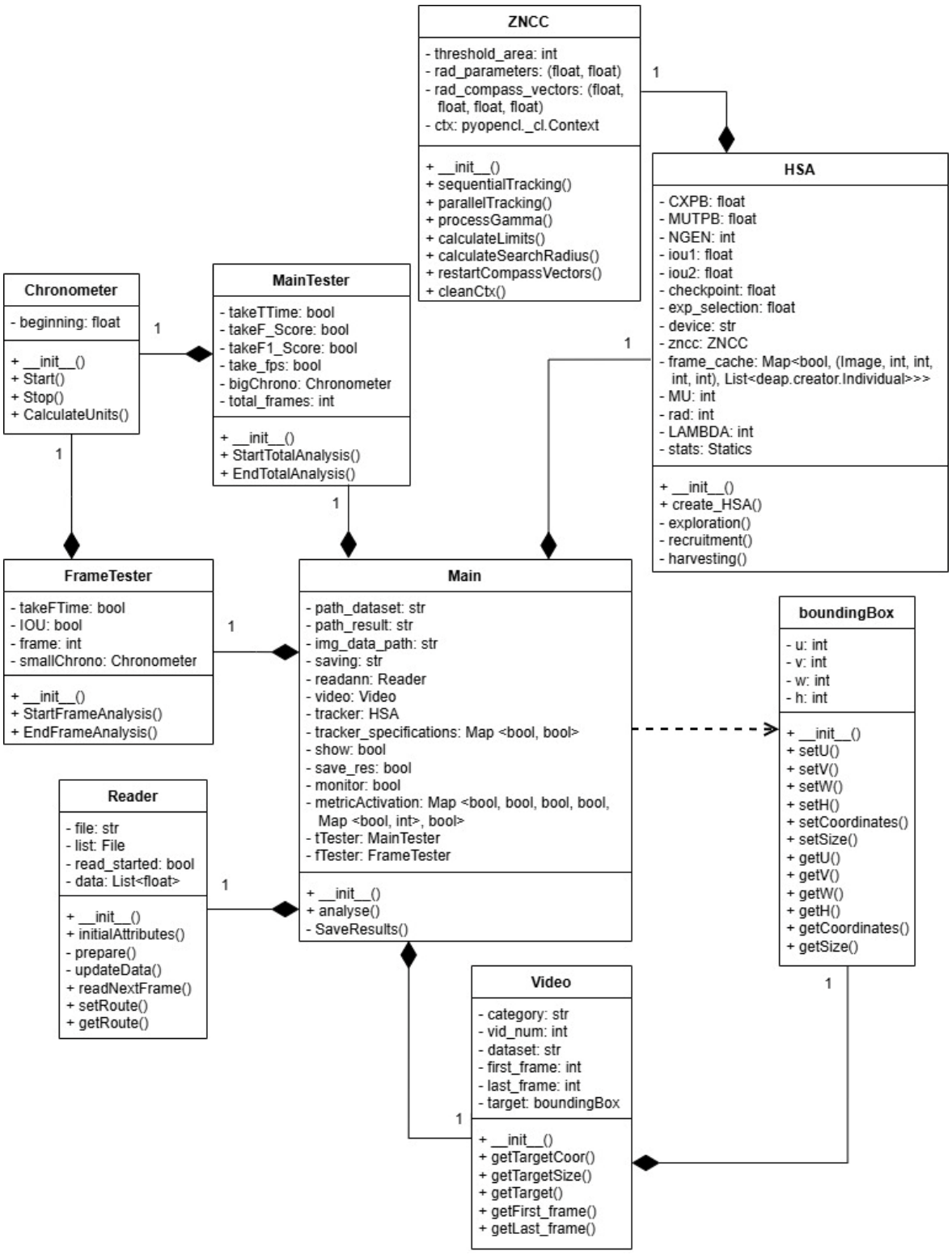

Figure 4.3. Class diagram for the OOP system part

Also, as can be seen, there are two test classes that belong to the Testing component: MainTester and FrameTester. MainTester class is the one that oversees carry out the complete video tests such as F-Score, FPS and total execution time, while FrameTester is the class in charge of carrying out the individual frame tests such as IoU and the execution time per frame. Both have an aggregation relationship with the Main class since both evaluations are merely optional, and the system can be executed without testing the metrics. To calculate general video metrics the information flow is the following: first, the individual frame metrics (such as IoU) are calculated by FrameTester class and frame by frame they are collected by the Main class, then, the Main class send those results to the MainTester class which process them in order to get the general final metrics (such as F-Score) once all the frames have passed. As can be seen, Chronometer class (which obviously represents the Chronometer component) has an aggregation relationship with the two tester classes because it is used to calculate metrics that can be disabled and not calculated if the user wants it in that way.

Another thing that can be seen in the class diagram is that not all components seen in the design diagram (Figure 4.1) are represented, this is because (as was previously said) not all the parts of the code were made following the OOP paradigm. This can be seen especially in the other metric components such as F-Score and FPS which have no classes and are composed solely of individual code functions that can be imported and called by other components and classes. E.g., Precision, Recall and IoU code are each composed of only one function and no more, and both functions are used by the functions of the F-Score component which is also used by the MainTester class.

About Testing component, as previously said, it has two classes that oversee the whole video evaluation and the individual sole frame evaluation, and which are called by the Main class. But the Testing component also contains a Main function that is used only when it is wanted to be executed independently as a system apart. This Main function employs both the MainTester and the FrameTester classes in order to process data coming from .csv files.

4.3 Parallel implementation of ZNCC algorithm

As was previously said in Chapter 3, ZNCC is a correlation image method that through a statistical calculation (Equation 3.1) gets the similarity rate between two images or regions of images (Figure 3.1) and it is used in VOT to compare different regions of an image frame with an image template of the target object that is looked for. Also, we take advantage of the parallel computing approach to process different regions of a frame at the same time (Figure 3.4).

In this implementation, ZNCC instances are parallelized by taking advantage of GPU cores. As it is described in Chapter 3, each GPU processing core executes a thread where a ZNCC instance is executed. All this process is managed by the *parallelTracking()* method of the ZNCC class (Figure 4.3). As was previously said in section 4.1, the OpenCL library for Python requires the creation of kernel (code to be parallelized) in C language. In ZNCC component, this kernel C code (Figure 4.5) is contained in the *parallelTracking()* method along the Python code (Figure 4.4) in charge of managing the threads including the processing of frame information before sending it to the kernel and processing the results that kernel sends after the ZNNC parallel work ends.

```
Convert template and frame into gray scale
Flatten frames into one-dimensional arrays
Calculate average template value
Initialize kernel arrays
Initialize OpenCL command queue
Build kernel program
Execute parallel ZNCC kernel code instances
Unpack ZNCC gamma results
Return gamma results
```

Figure 4.4. Pseudocode of the Python section of the ZNCC component

```
Tbar = average template value
Ibar = 0
For each pixel (x, y) in frame region:
      Ibar += pixel (x, y)
Ibar /= region area
Sum1 = 0
Sum2 = 0
Sum3 = 0
For each pixel (x1, y1) in frame region and each pixel (x2, y2) in
template:
      Sum1 += (pixel (x1, y1) - Ibar) * (pixel (x2, y2) - Tbar)
      Sum2 += (pixel (x1, y1) - Ibar)^2
      Sum3 += (pixel (x2, y2) - Tbar)^2
Gamma = Sum1 / Square root (Sum2 * Sum3)
Return Gamma
```

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

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

$$\sum_{x,y}\left[I(x,y) - \bar{I}_{u,v}\right]\left[t(x-u, y-v) - \bar{t}\right]$$

$$\sum_{x,y}\left[I(x,y) - \bar{I}_{u,v}\right]^2$$

$$\sum_{x,y}\left[t(x-u, y-v)\right]^2$$

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

Figure 4.5. Pseudocode of the ZNCC kernel code and how it is related to the ZNCC formula (Equations 3.1 – 3.3)

It is worth mentioning (and as it was established in the foundations on GPU parallelization in Chapter 3) that since the kernel code is parallelized in a GPU, all the threads belong to one single work block. This allows us to establish a global memory to share information among all the threads and the Python code, information such as the frame itself and template matrices, the average template value (the only ZNCC equation part that is calculated in the Python section outside the kernel), the frame size, the template size, and a gamma array, the only piece of global memory that can be written by the threads because here is where the gamma results of each thread is stored and eventually read by the Python code. In its basic form capable of being used as a single tracker (without the HSA presence), the ZNCC component also selects the best gamma result by itself, but in its final form adapted to be used along the HSA as its fitness function, the ZNCC component returns all gamma results in order to be processed by the HSA component and its evolutionary-swarm mechanisms (Figure 4.6).

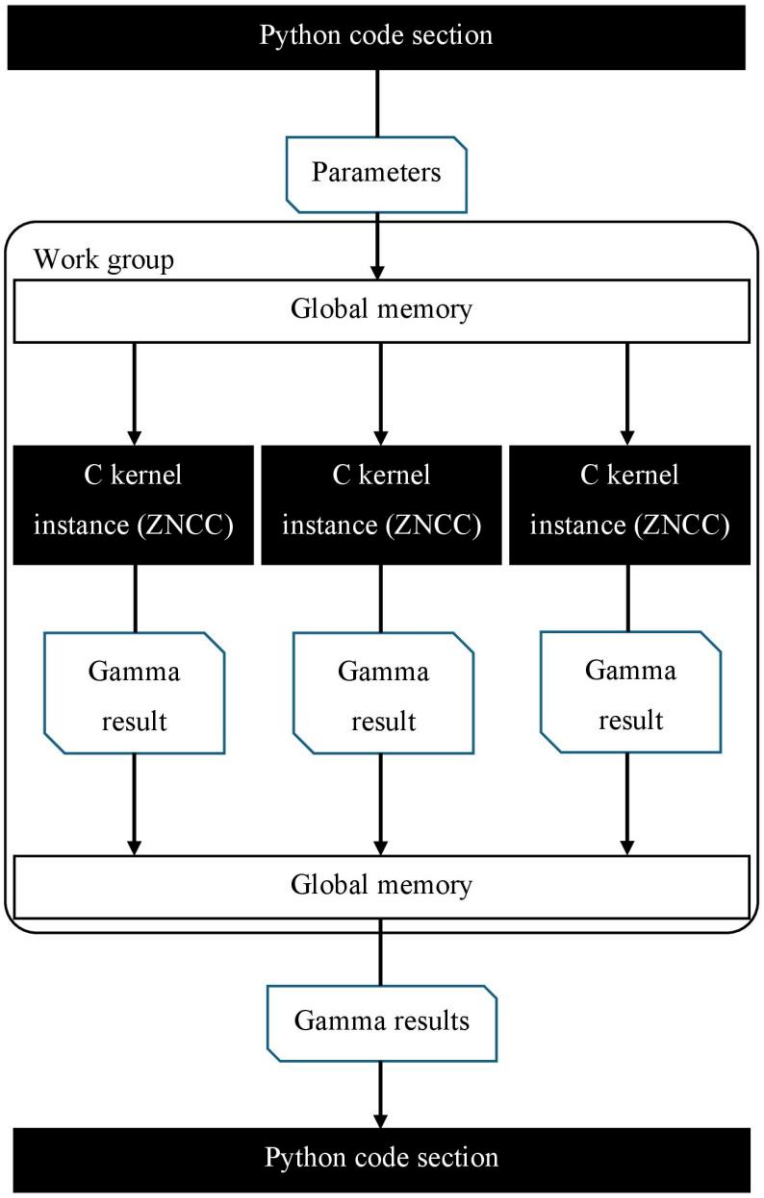

Figure 4.6. The Python code section sends the necessary parameters to the parallel section work group's global memory that is accessed by the C kernel instances. When the parallel kernels finish their job, they send their gamma results to the global memory that is accessed by the Python section who process the gammas

It is worth mentioning that, on previous VOT ZNCC implementations [15], the average template value was calculated inside the kernel, not outside. Since this is the same single value used for all threads, in this new improvement, the average template value is now calculated outside the threads and before the kernel execution in order to calculate it

only once and not spend resources calculating it many times unnecessarily inside the threads. This is one of various improvements that have been implemented in this new version.

Another improvement that does affect the system's performance in a bigger and more positive way was to establish only a determined percentage of the pixel areas to be processed instead of 100% of them. One of the observations that were made about how the ZNCC algorithm is executed in parallel and it could be optimized, is that in the previous approach an exhaustive search was done by executing ZNCC on all possible areas of the frame based on 100% of the pixels, which could generate redundancies when analyzing different areas of the frame that were very similar to each other and therefore, unnecessary computational cost was consumed (because of the creation of unnecessary threads) which would be the cause of memory overflow and the incapability of certain videos (especially those with higher resolution) of being able to be analyzed. Based on this observation, a hypothesis was formulated that, if we work on a percentage less than 100% of the total number of pixels in the frame, redundancies could be eliminated by the use of a correct implementation carried out with a percentage of pixels that were not so low (20%, for example) and at the same time distributed randomly and uniformly throughout the frame total area.

Following the logic that it's impossible for an object to have drastic movements at a large distance between one frame of the video and the next, a reduction in the total object search area was also implemented (Figure 4.7). With this new improvement, instead of searching within the entire frame, only a small area around the object's last position is searched, which is where the object will likely be found in the new frame.

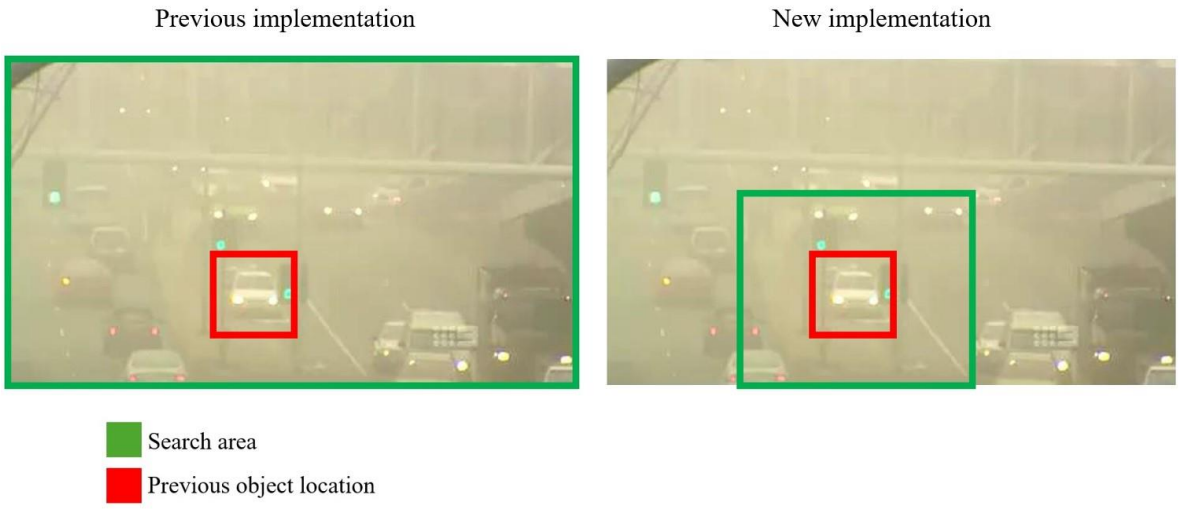

Figure 4.7. Change on the search area with the new reduction improvement

This new improvement aims to reduce the processing required for each frame by eliminating unnecessary searches, while also increasing the probability of success by focusing on the search area where the object must be located. Originally the reduction in the search area was set to a perimeter of 2 times the size of the object around it, however, this later changed so that reduction change could be customizable and adaptable to a preset percentage in relation to the size of the frame (Figure 4.8). This was later changed to instead of just being based on a single percentage value, it would be based on four, one for each side of the search area (Figure 4.9).

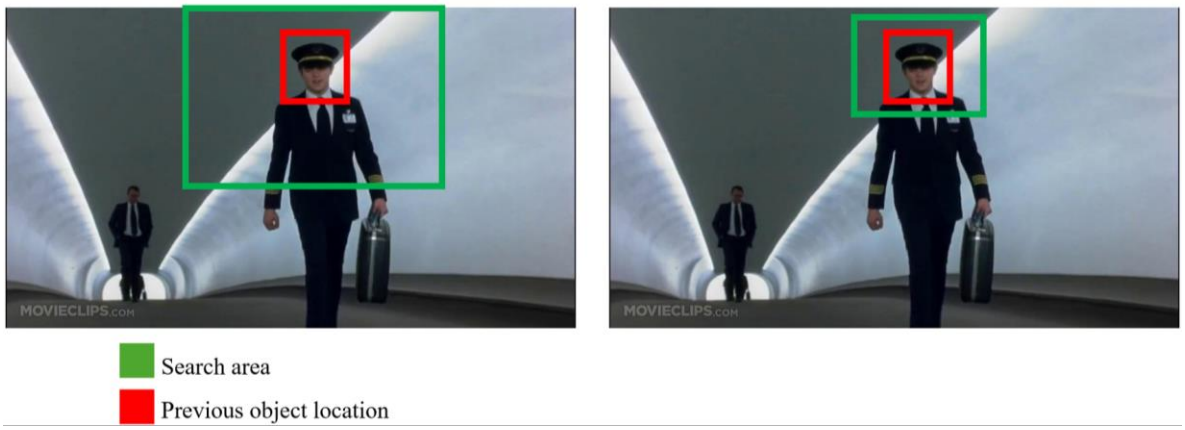

Figure 4.8. Change on the search area with predetermined percentage based on the frame size (50% and 75%)

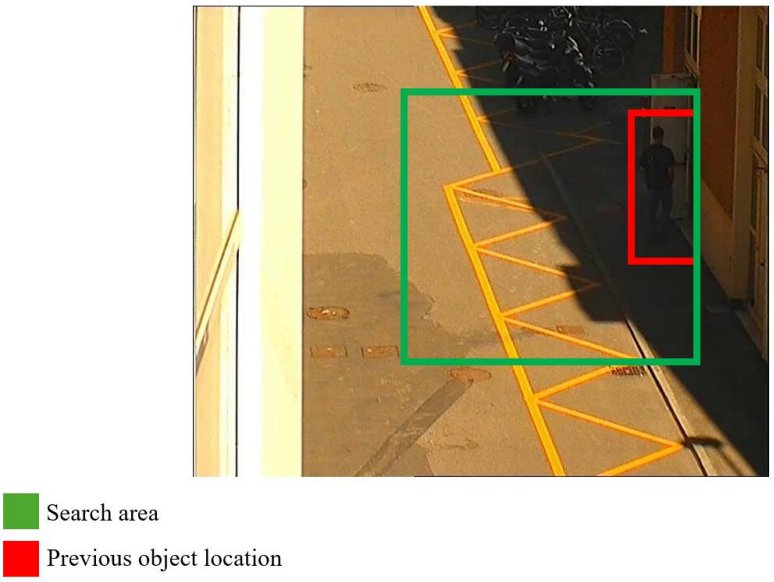

Figure 4.9. Change on the search area with the four predetermined percentages (75% up, 50% down, 50% left, 99% right)

This improvement was implemented considering these percentages would be granted by the HSA algorithm's swarm intelligence which would indicate the sides towards which the object is most likely to move within the video and therefore where the search should focus more. However, by basing these percentages on the frame size, there would be a risk of retake the previous observed failure of analyzing extreme sections of the frame where the object is very likely not to be (for example, if the search percentage is set to 100% on one of the sides), for this reason, it was decided to delimit the maximum of the search area to a radius of 3 times the object size (Figure 4.10).

Figure 4.10. Change on the search area with the four predetermined percentages based on 3 times the object size radius (75% up, 50% down, 50% left, 99% right)

This version of the moldable search area improvement proved to have good results, but it was still not perfect so, based on some investigations that mention factors to consider about the definition of search area in VOT [1] [8] [82] [83] [84] [85] [86] the next formula (Equation 4.1) was created in order to calculate the best search radius considering both the object size (bounding box) and the frame size:

$$r = \min\left(\propto \cdot \max(w_b, h_b) + \beta \cdot \sqrt{W^2 + H^2}, \frac{\min(W, H)}{2}\right) \quad (4.1)$$

where w_b, h_b are the bounding box's width and height, W and H are the frame's width and height, and the α and β values are the parameters that define the weight that the object's size and frame's resolution will respectively have in the result. In case of the α parameter, it represents the area in which the object is believed to be able to move relative to its size, for example, if α = 2 it's because we think the object could move thorough a radius twice its size. The β parameter is more delicate since it adds extra search area considering the total frame resolution, or (more specifically) the maximum linear distance that an object can move across the frame area ($\sqrt{W^2 + H^2}$ = the Euclidean distance between one frame corner and the other), so it is recommended to keep it on range of 0.0-0.1 but it can be bigger

according to the system's needs, especially if the object moves fast and it can make huge position jumps between frames.

As can be seen, the equation to calculate the radius is essentially composed by two formulas ($\propto \cdot \max(w_b, h_b) + \beta \cdot \sqrt{W^2 + H^2}$ and $\frac{\min(W,H)}{2}$) from which the minimum result is extracted. This ensures that the search area not exceeds the frame's size (represented by the second formula), but it can generate a search radius that sometimes could exceed frame's limits depending on the position of the object, so the search area is later adapted to not generate a computing error (because of trying to read pixels that doesn't exists outside the frame). That's why sometimes it doesn't cover the area that it should but instead covers an area that reaches the frame's limits (Figure 4.11).

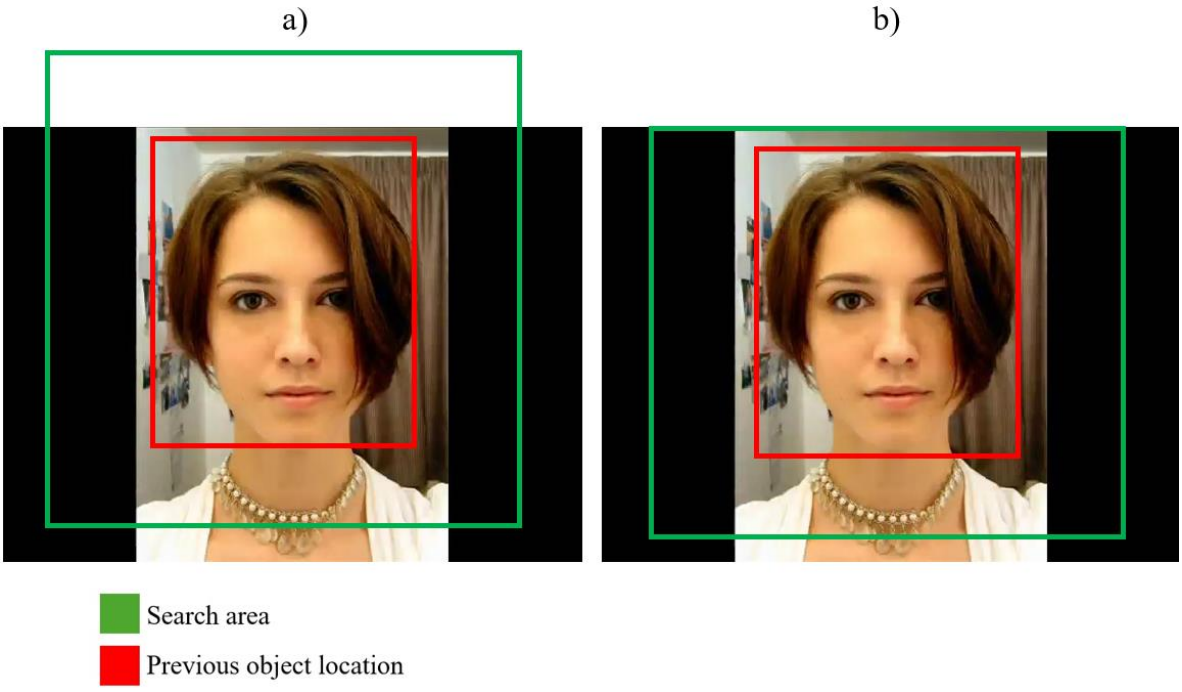

Figure 4.11. a) The search radio that was calculated surpasses frame's limits, b) Search area is adjusted to stay in the frame's limits

Another improvement that was applied is the “blinds method”. This is an original implementation based on the idea that an object can still be distinguished even if there is low interference in the image. A change was implemented in the ZNCC algorithm code to the cycles that traverse the image pixels, so instead of moving pixel by pixel, a jump of two pixels is made and the system only process half of the frame's information (like watching the image pass through blinds) (Figure 4.12).

Figure 4.12. The idea behind the blinds method. a) Original frame, b) same frame after applying the blinds method

Although the mathematical results of the ZNCC equation (Equation 3.1) vary when removing half of the pixels from the analysis, when applying the same blinds algorithm to both frames (the current and the previous one), the statistical relationship between these two is maintained, something that has been corroborated in experiments and tests where it is observed that there is no noticeable decrease in the effectiveness of the ZNCC algorithm tracking the object when this method is applied. It is worth mentioning that, even if this method deletes certain amount of image data from the frames in order to reduce computational efforts, it is not exactly the same as dawn-sampling [87] since the blinds method does not reduce frame’s resolution and just omits information, unlike down-sampling which is based exclusively on reducing the frame resolution to generate a new version of the image that is easier to process. We could say that in a certain way the blinds method is a variant of down-sampling, but very far removed from its traditional methodology.

4.4 Honeybee swarm video tracking system

4.4.1 Honeybee swarm algorithm proposal

This new HSA implementation maintains the three basic phases of the algorithm that are based on the natural process that bees use to search and collect food (exploration, recruitment, and harvesting) but trying to carry them out in a more optimal way instead of the previous approach that is described in Chapter 2. Unlike the previous implementation that applied the three HSA phases in each single frame (Figure 2.15), the new implementation looks for using the three phases strategically using the exploration phase only in the first frame to locate the object, while in subsequent frames are only used the recruitment and harvesting phases. The idea is that exploration bees must find the locations where the object is most likely to be found, then the recruitment phase assigns harvester bees near the locations of the explorer bees with the high fitness score to then be released and concentrated in those locations in the harvesting phase (Figure 4.13).

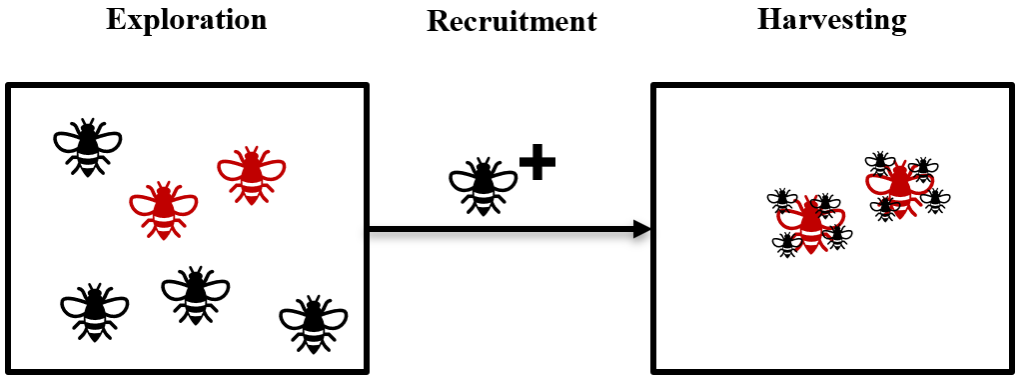

Figure 4.13. Bees look for the best positions (exploration), new bees are assigned to those positions (recruitment) and then they exploit them (harvesting)

The exploration phase is applied again only if the target is lost, to identify if this happens, frame results are evaluated using IoU after processing the frame and it is compared to a predefined threshold, if IoU is under that threshold, then the target is lost. It is worth mentioning that IoU is not evaluated after every frame, and this depends on how the system is configured and what is the next available ground truth. In the case of this

research, ground truth availability depends on the ALOV300++ annotation files in most of which the ground truth is written but every five frames for each video.

There are two types of exploration: simple exploration and exhaustive exploration. Simple exploration is the normal HSA exploration that looks for the object in a certain limited search area (that is calculated based on the frame's size and object's size, as it was explained in section 4.3) using a population of bees that evolves in a certain number of generations through an evolutionary strategy that in this implementation is the $\mu + \lambda$ strategy (consult Chapter 2 for more information about evolutionary strategies). On the other hand, exhaustive exploration does not employ any evolutionary and swarm process since it does an exploration on the entire frame zone (Figure 4.14). Another difference between these two types of exploration is that (as was previously said) simple exploration deploys a predetermined number of bees which is the population size that was chosen for the HSA execution of that particular video, to choose this population size there is a mechanism that will be explained later, but the difference with the exhaustive exploration is that in this exploration the number of pixel zones that are processed is not related to the size of the bee population; instead, a percentage of the total pixel size of the frame is taken similar to how it is explained in the section 4.3.

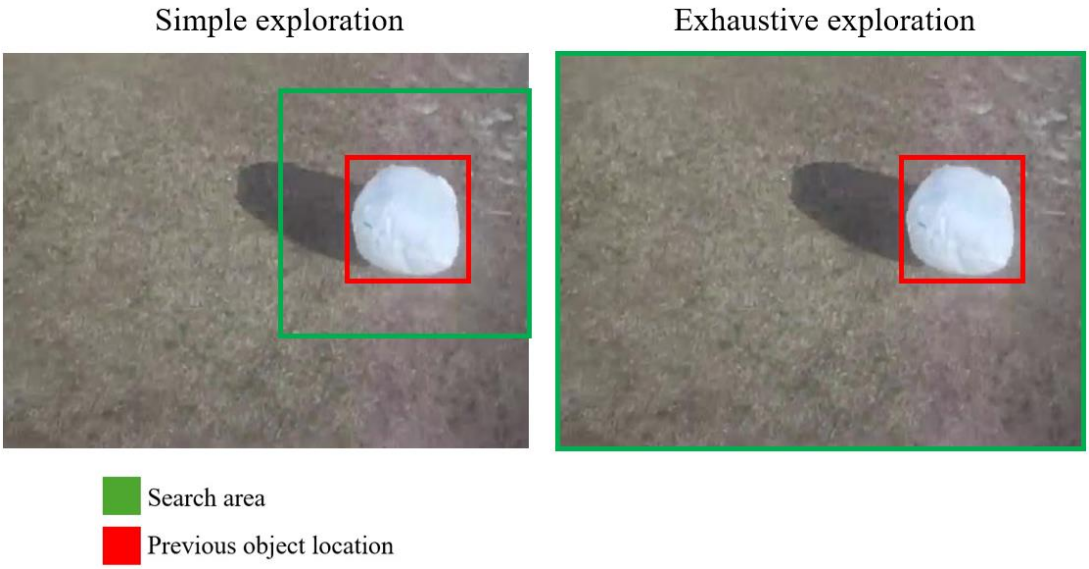

Figure 4.14. In simple exploration the search area is reduced covering only a part of the frame, while in exhaustive exploration the search area covers the entire frame

As was said before, simple exploration is used in the first frame (primarily to initialize the evolutionary algorithm) and eventually only if the IoU evaluation decreases under certain threshold that has been established as 0.5 in the final configuration (after different adjustment tests whose results can be seen in the Appendix 1), which means that the object is being lost but is close to the actual position (more information about IoU meaning will be explained in section 4.5). Otherwise, if IoU = 0.0 means that there is no intersection between the truth and the ground truth, so the object is lost and it could be close to the actual position, but it can be far too, so that is when the exhaustive exploration is used. Both simple and exhaustive exploration are used only in the frame after the frame when the IoU evaluation crossed the threshold necessary for its respective activation, after that, the following frames execute only the recruitment and harvesting phases until IoU crosses those thresholds again (Figure 4.15). In cases where exploration will not be performed in the current frame nor was it performed in the previous one (so there are not explorer bees as such), new harvester bees will be assigned in the recruitment phase to the best harvester bees from the previous frame.

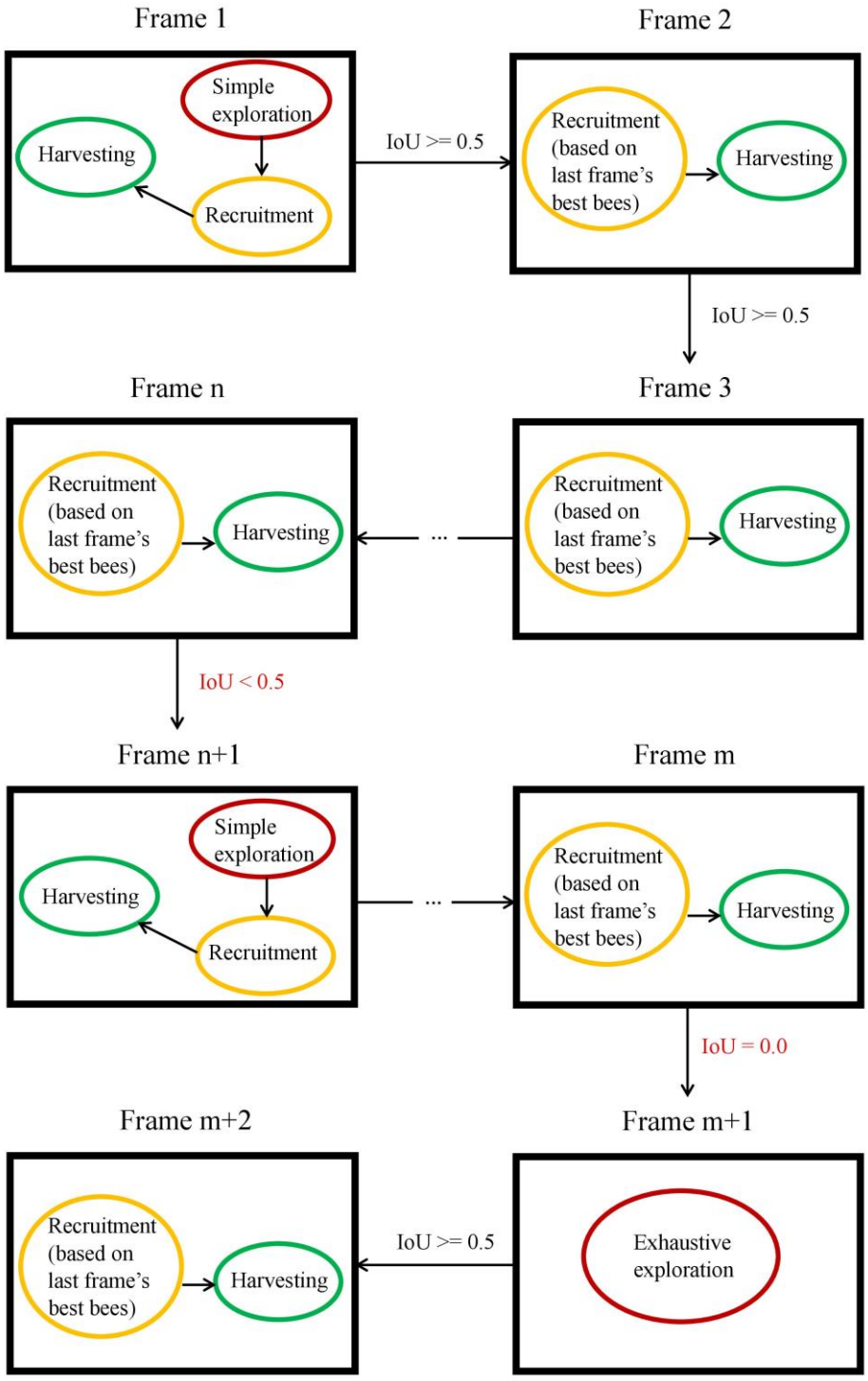

Figure 4.15. In the first frame, the three HSA phases are executed (including the simple exploration), and in the following frames just the recruitment and harvesting phases are executed until IoU is under the first predetermined exploration threshold. If IoU = 0.0 then the exhaustive exploration is executed to continue with the same process

The previously mentioned detail of evaluation being made after a certain number of frames (2 – 10 in the ALOV300++ benchmark) instead after each frame, makes a constant single frame evaluation impossible but also helps us to introduce an improvement called "checkpoint frame". As was explained before, ZNCC requires an object template that is normally extracted from the previous frame, this creates a disadvantage because if the system detects by mistake something that is not the target, then it will search the incorrect object in the frame based on an incorrect template and if the template is not eventually corrected then he object position may be lost forever until video ends. To prevent this, when IoU is checked and it is still over certain predetermined threshold (that is different from the previously mentioned threshold that executes simple exploration) the template extracted from that frame is saved (this is the checkpoint frame), then when the object is lost, IoU decreases and is time to execute any of the two types of exploration, the template that is taken as reference to the ZNCC is that checkpoint frame that was saved (Figure 4.16). The IoU threshold that activates the checkpoint frame saving was established as IoU = 0.6 in the final configuration (this as the results of some adjustment tests that will be explained in Chapter 5).

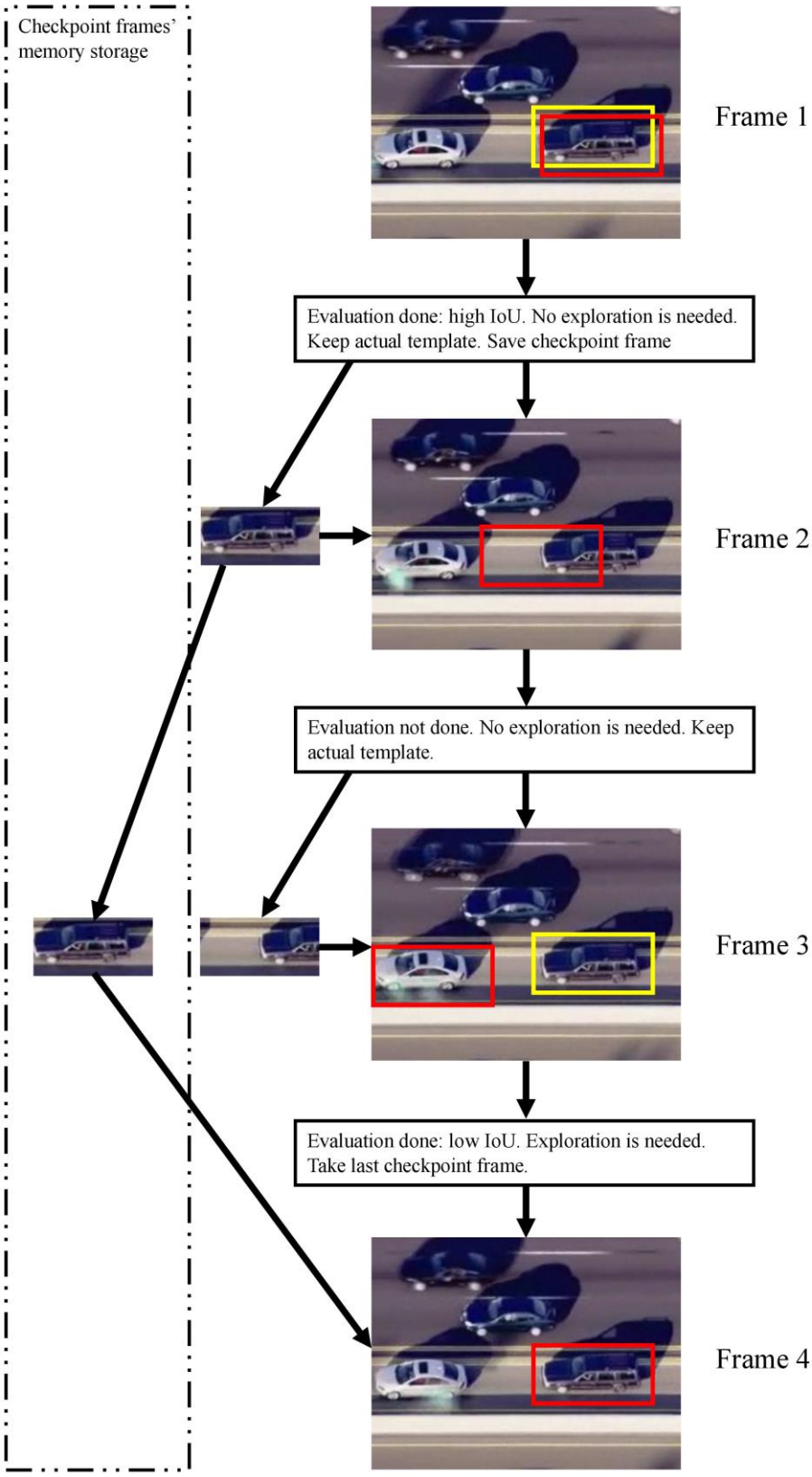

Figure 4.16. Example of how checkpoint frame works. Frame 1 gets a high IoU evaluation (red square=truth, yellow square=ground truth), so the resulting template is saved as checkpoint frame. Frame 2 isn't evaluated. Frame 3 gets a low IoU evaluation, so its resulting template is discarded, and last checkpoint frame is used in Frame 4

Regarding how the bee population is assigned, this number is chosen based on the search radius that is calculated for that particular video as it is explained in the formula shown in section 4.3 (Equation 4.1). After calculating the search radio for all the ALOV300++ videos and analyze all of them, a six-category system was created (Table 4.2) considering the statistical range in which the search radii fluctuate and different tests performed using different population sizes (Appendix 2), it is worth mentioning that it is advisable to use population sizes that are powers of two (population = 2^n).

Radius range	Population size
0 - 111	512
112 - 197	1024
198 - 283	2048
284 - 368	4096
369 - 454	8192
455 - 540	16384

Table 4.2. Population sizes according to the possible radius sizes that ALOV300++ can have

Getting into the three phases specifically working, the three keep the general evolutionary process that was described in Chapter 2 following the μ + λ strategy, especially in exploration and harvesting phases which are the only two that use evolutionary processes. In the case of exploration phase, first an initial μ population is created and evaluated (using the ZNCC algorithm), then the λ population is created using selection, crossover and mutation based on the best results from μ population. As was said in Chapter 2, these three operations are the principal reproduction operators (along copy which we don't use) and to carry them out in this implementation, different methods were tested and chosen according to their results (Appendix 3). The following list describes the methods that were tested and chosen:

1. **Selection –** Tested methods: roulette and tournament. Chosen method: tournament (four individuals per tournament).

2. **Crossover –** Tested methods: uniform crossover, blend crossover and SBX. Chosen method: blend crossover.
3. **Mutation –** Tested methods: uniform mutation, polynomial mutation and gaussian mutation. Chosen method: gaussian mutation (σ = 15, mutation probability = 0.3).
4. **Copy –** it is not used in this implementation.

After the λ population is created, it is evaluated to merge its best individuals with μ's and create a new μ population to be used in a new generation. This process is repeated in every generation until the best explorers are defined.

In recruitment phase, harvester bees are assigned to each one of the best explorer bees' zones thought an exponential distribution described by the following equation (Equation 4.2):

$$r = b(1 - e^{-x}) \tag{4.2}$$

where *r* is the number of harvester bees assigned to each explorer, *x* is the fitness score of that bee and *b* is the number of bees that we have assigned them. In this way, we give the harvester bees to the explorer bees according to how high their fitness score was. It is worth mentioning that the population of harvester bees is half of the explorers, but also a new mechanism was included to "turn" the failed or unused explorer bees into harvester bees, so the number of low score explorers that didn't get any harvester bees is added to the harvester's population (Figure 4.17).

```
For each explorer bee until harvester population == 0:
      Recruits = harvester population * (1 – e^(explorer bee's
      fitness score))
      Assing recruits to explorer bee
      Harvester population -= recruits
If harvester population > 0:
      Assing all harvester population to best explorer bee
Harvester population += explorer bee with 0 recruits
For each explorer bee until harvester population == 0:
```

```
        Recruits = harvester population * (1 – e^(explorer bee's
        fitness score))
        Assing recruits to explorer bee
        Harvester population -= recruits
If harvester population > 0:
        Assing all harvester population to best explorer bee
```

Figure 4.17. Pseudocode of HSA

The harvesting phase follows the same process as the exploration phase but applied to each group of harvester bees that was created around each explorer bee. The same evolutionary operators that are used in exploration phases are conserved in harvesting phase, and (as was said earlier) the best harvester bees are saves to be used as "explorers" to which we are going to assign new harvesters in the next frame, except if in the next frame we have to execute the exploration again, in that case new explorers will be generated.

The chosen population size is fixed and cannot change during video execution, except in one case. During preliminary testing of the latest code version, it was discovered that some videos in the ALOV dataset were still failing with 0% IoU from the first frame even with a maximum number of bees. Considering these videos to be lost causes, a mechanism was included to reduce the bee population to 16 bees after reaching a certain number of frames with a 0% IoU. This allowed the videos to finish as quickly as possible without wasting computational resources.

4.4.2 Parallel system implementation

As was previously said, harvesting phase basically consists of replicating the same evolutionary process that was previously done in exploration phase but on smaller bee groups. To optimize this (and as was briefly explained in section 4.1) we took advantage of Joblib, a parallel computer Python library compatible with the DEAP evolutionary library. Joblib allowed us to parallelize the harvesting phase in order to process all the previously mentioned sub-populations at the same time in a parallel way (Figure 4.18). It is worth

mentioning that due to the Joblib's limitations, all the process parallelized by this library are executed into CPU threads, except for the ZNCC evaluation which are still parallelized on OpenCL and into GPU threads.

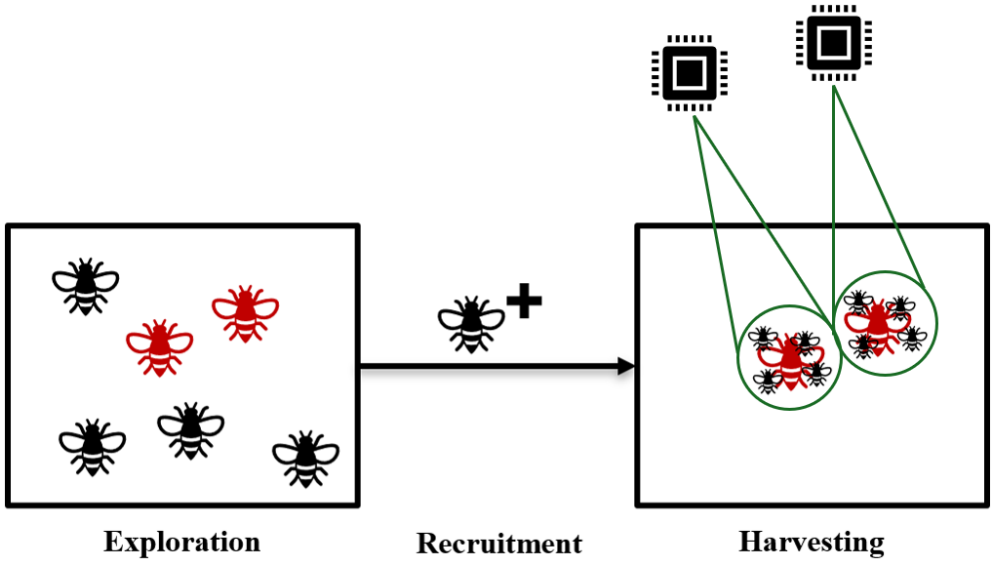

Figure 4.18. In harvesting phase, each group created around the explorer bees are assigned to a CPU processor so they can be processed in parallel

With all the parallel and sequential phases of the HSA and ZNCC algorithms described, and how these relate to each other, the system's general mapping (or at least the VOT part and its distribution between the CPU and the GPU) would be as follows (Figure 4.19):

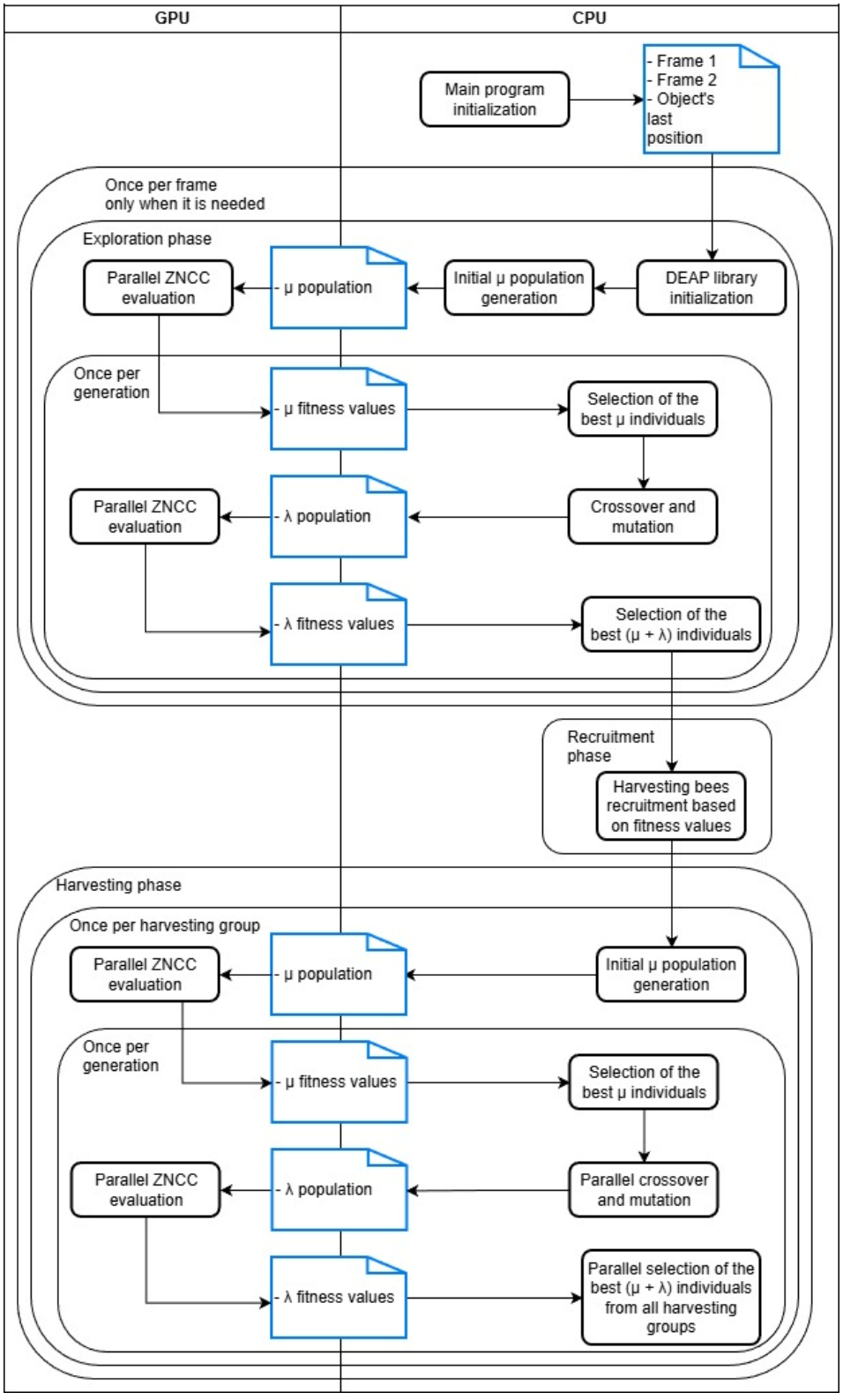

Figure 4.19. General task distribution between the CPU and GPU

4.4.3 Problems found

Different problems and obstacles came up when the system was in the development process. One of the obstacles that consumed developing time was the "multi-kernel" improvement, an idea that could improve the parallelization of the system's ZNCC part but instead slowing down the system speed without representing a significant improvement.

The multi-kernel improvement sought to carry out the algorithm through the collaborative work of three kernels in charge of managing, optimizing and executing the ZNCC work. The main innovation behind this new approach was how it seeks to take advantage of the processing individual pixels through a more efficient parallel approach, in which an area is analyzed along with the four surrounding areas based on 4 positive and negative displacement values on the X and Y axes (Figure 4.20).

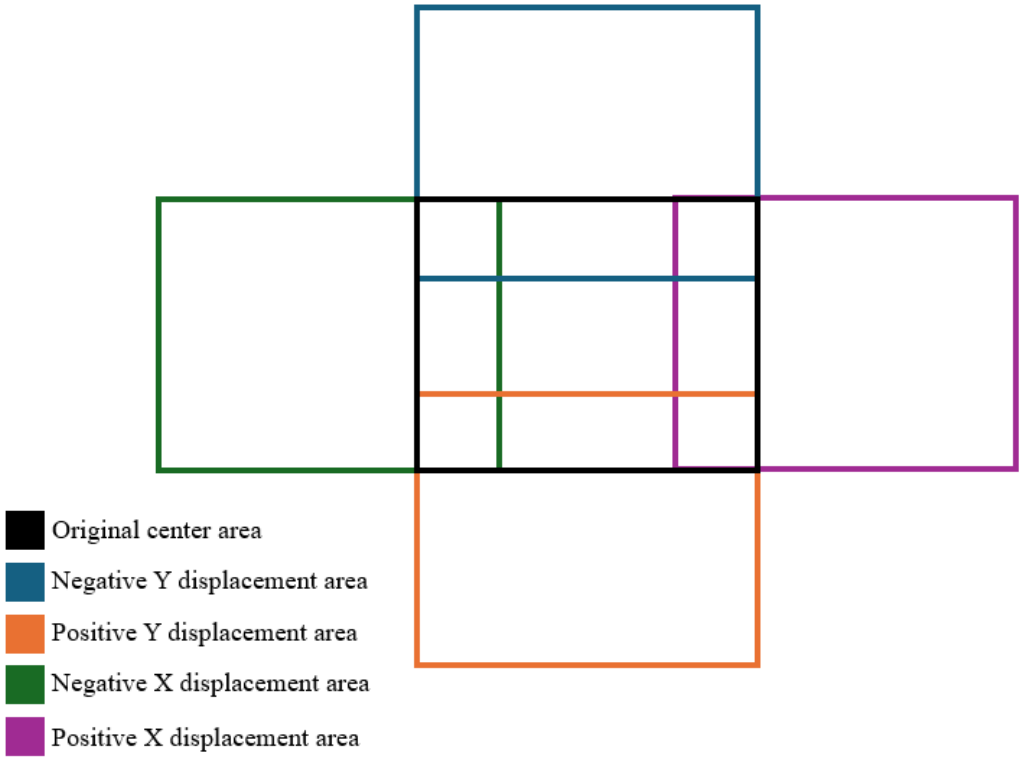

Figure 4.20. Analysis area with its four surrounding areas in the discarded multi-kernel improvement

This parallel approach focused on how the pixel path and its summation are leveraged to carry out the initial calculations of the next ZNCC equation (Equation 3.1). In this new approach, a path is made of all the pixels that compose the total area resulting from the overlap of the central area and its surrounding area, taking the path and summation of the pixels that compose the overlap zones between the different areas (Figure 4.21) and leveraging them to obtain the summation value of the numerator of the $\bar{I}$ value (Equation 3.3) of the ZNCC equation instead of making the path more times and falling into redundancy.

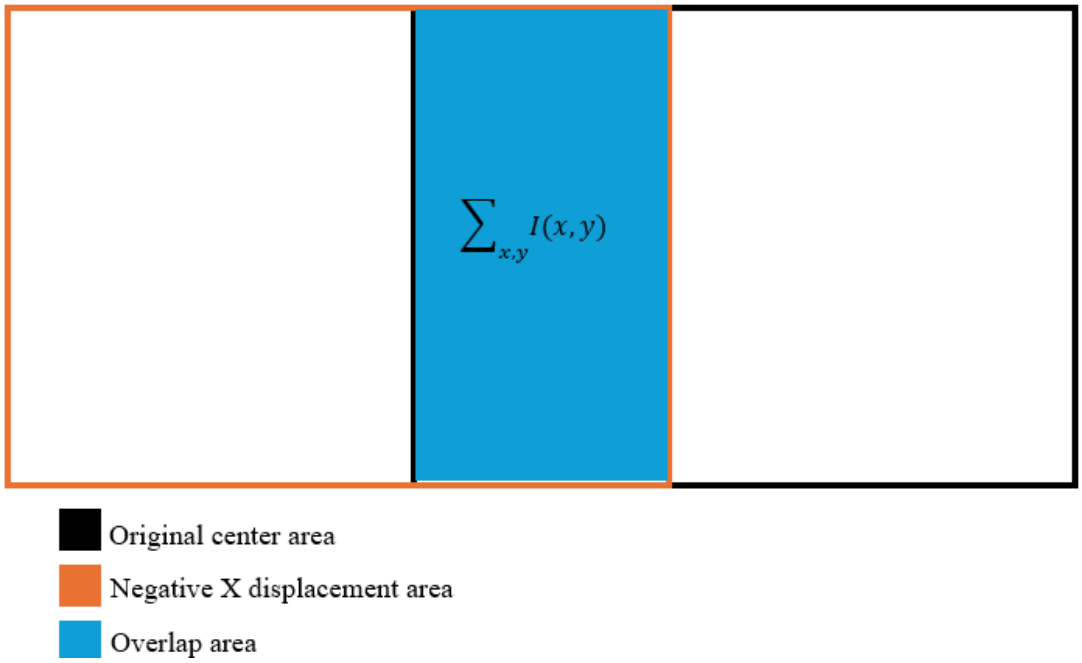

Figure 4.21. Overlapping areas

With this new approach, we needed to create 3 new kernels to be executed on GPU:

1 Area administration kernel: which would calculate the coordinates and size of the overlap areas created by the overlap values (which would have been provided by the HSA algorithm in the future).

2 Sub-area calculation kernel: which calculates the sum (s) of the pixels that compose each overlap area, following the next equation (Equation 4.3):

$$s_i = \sum_{x,y} I(x,y) \tag{4.3}$$

where i is the number of the overlap area.

3 ZNCC final calculation: this kernel would have obtained the final ZNCC result of each analysis area getting the total sum of all the overlap areas that compose it to obtain the $\bar{I}$ (Equation 4.4) value specific to that area and then calculating the complete ZNCC value (Equation 3.1).

$$\bar{I}_{u,v} = \frac{\sum_{x,y} s_i}{m \times n} \tag{4.4}$$

All three kernels for this version of ZNCC were completed but tests indicate that this methodology fails to achieve the significant improvement it was expected, instead presenting very similar (or even worse) results than the traditional ZNCC algorithm single-kernel version, and entails significant resource demands during data transfers between kernels. Therefore, it was decided to definitively discard the use of this methodology although in future work it could be reused (perhaps not completely, but certain useful parts).

But multi-kernel was not the only improvement that consumed developing time. Another improvement that was discarded is the “bubble compass”, an improvement that looks for adjust the search area according to how the object is moving. With the bubble compass, a special method of the ZNCC class receives the object position on the current frame and on the last one and calculates the object's movement vector to increase the search area on the sides where the object tends to move and decrease it on the sides where it is not moving (Figure 4.22). With this approach, the bees "follow" the target more effectively, and concentrate on areas where it is more likely to be found.

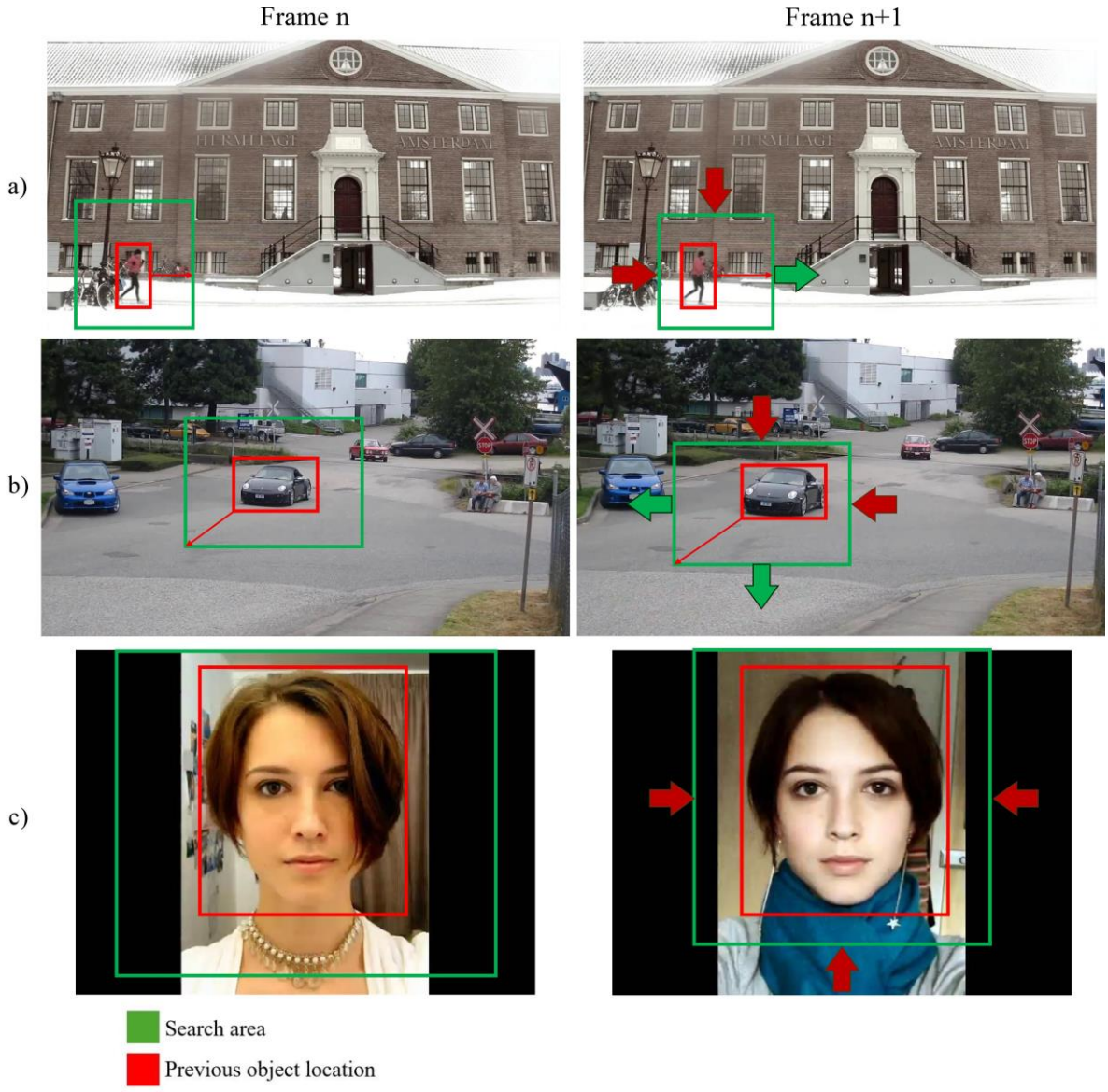

Figure 4.22. Bubble compass modifies the search area: a) the object moves to the right so right side increases while top side and left side decreases, the bottom side is not affected since it is already reduced to not overpass the frame's limit, b) the object moves diagonally down and to the left, so left side and bottom side are increased while top side and right side are decreased, c) the object does not move at all so all sides are decreased

Thos first implementation of the bubble compass proved to work well on this implementation but although it did not drastically affect system performance, it did slightly reduces the system's ability to successfully track the target, so it was decided to disable it

and just like the multi-kernel improvement, we believe that a correct implementation of the bubble compass could be truly effective and beneficial in future work.

During the development, other obstacles were noticed related to the disadvantages of how the Open-CL library is used and how it has been used in previous implementations. An analysis of the time taken by each section of the ZNCC code showed that the memory cleanup allocated to the OpenCL kernels for the ZNCC algorithm sometimes took longer than the algorithm itself. An investigation into the operation of the OpenCL library led to the conclusion that this is because OpenCL's cleanup functions do not operate in parallel but rather sequentially traverse memory spaces for cleanup. It was also later discovered that manual memory cleanup from the code was not necessary, as the OpenCL Python version has a memory cleaner linked to Python's own cleaner which automatically disposes of all that memory. These cleanup functions were removed showing no effect on the effectiveness of the code but a positive effect making the ZNCC algorithm faster, something that especially benefited multi-kernel versions which accumulate a lot of memory due to the exchange of data between kernels.

Further analysis of the timing of the ZNCC component showed that something that also consumed considerable time (especially in lightweight videos) was the creation of the OpenCL context. Running OpenCL library requires the creation of a specialized environment called "context," a sort of digital container where the kernels and their shared memory are stored. Timing analysis showed that creating this takes approximately 100 ms per frame in all videos, which was an obstacle in the way to our goal of reaching 15 FPS as this would require the system to have a maximum frame rate of 66 ms. To solve this problem, it was decided to create a single context which would be created at the beginning of the video processing as a ZNCC class attribute which would be used for all frames eliminating the extra 100 ms in each frame. Also, this improvement does not cause a buffer overflow thanks to the OpenCL memory cleaner which works automatically.

About the use of DEAP library, while studying and analyzing the previous implementation code [15], it was observed that the HSA code of that implementation worked through 6 parameters that allowed controlling the percentages of offspring bees

generated through crossover, mutation and random in both the exploration and the harvesting phase of the HSA algorithm. As mentioned above, for this implementation the specialized DEAP library for evolutionary computing for Python is being used. Within this library there is a special function called *varAnd()* which allows generating an offspring population from a parent population by automatically executing the crossover and mutation operators. However, it does not allow manual control of the percentage of offspring obtained with both operators, nor does it allow generating random operators. To replicate the previous implementation manual approach, a custom function was created to replace DEAP's varAnd function. However, while this function did not demonstrate a decrease in F1-Score, it did not increase either and it was observed that it decreases speed in terms of FPS. For this reason, it was decided to abandon the use of manually configured percentages as previous implementation did and instead allow them to be automatically configured using DEAP's varAnd function concentrating the algorithm's tuning on parameters that DEAP does support, such as the crossover and mutation probability indices.

4.5 System evaluation

4.5.1 Benchmark dataset selection

As has been said, the ALOV300++ dataset [11] has been chosen as the evaluation benchmark of the system because of its video variety. The dataset is composed of 314 different videos that come from different sources and represent different challenges, all of them already divided into individual frames that are .jpg images. The different categories of the dataset and how many videos each one have can be seen in the following table (Table 4.3):

Category	Number of videos	Description
01-Light	33	Excessive light and/or sudden changes in light.
02-SurfaceCover	15	Changes in color and texture.
03-Specularity	18	Shiny and/or reflective objects.

04-Transparency	20	Transparent objects.
05-Shape	24	Changes in the shape of the target.
06-MotionSmoothness	22	Slow movements or absence of movement.
07-MotionCoherence	12	Erratic and unpredictable movements.
08-Clutter	15	Objects that follow similar movement patterns.
09-Confusion	37	Objects with a similar appearance.
10-LowContrast	23	Low contrast between the target and the background.
11-Occlusion	34	The view of the object is obstructed.
12-MovingCamera	22	Camera movement.
13-ZoomingCamera	29	The camera zooms in or out.
14-LongDuration	10	Long-form videos.

Table 4.3. Video category of the ALOV300++ dataset and the number of videos that each one contains

To evaluate the system (and as was briefly explained in section 4.2), the dataset contains a .ann file for each video where the location of the object frame by frame (a.k.a ground truth) is written. Each file is essentially a list of rows; each row represents one frame information and has nine numerical data where the first number represents the frame number and the other eight represent the coordinates of the object's bounding box, each of which is written twice following the next schema:

- Top left corner:
  - x coordinate: fourth and sixth numbers.
  - y coordinate: third and fifth numbers.
- Bottom right corner:
  - x coordinate: second and eighth numbers.
  - y coordinate: seventh and ninth numbers.

Also, it is worth noting that (as was mentioned earlier in this chapter) the .ann files don't have the information of all the frames of the video (Figure 4.21).

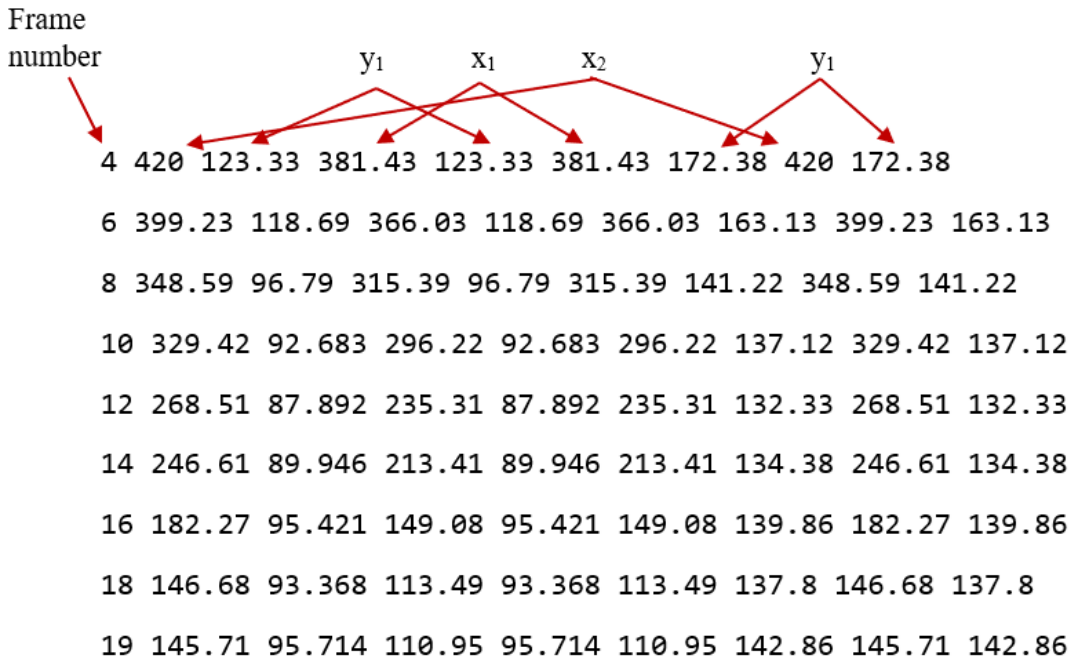

Figure 4.23. Example of a .ann file and its structure. (x_1, y_1): top left corner coordinates, (x_2, y_2): bottom right corner coordinates

4.5.2 Evaluation metrics

As has been told in this chapter and in previous chapters, the principal metrics to be used to test the system are F1-Score and FPS.

The frames per second (FPS) measurement is what we are going to use to measure the speed of the system to evaluate how many frames the system is capable of analyzing in one second of execution and it can be calculated using the next formula (Equation 4.5):

$$FPS = \frac{total\ video\ frames}{total\ execution\ seconds} \tag{4.5}$$

As was explained in section 4.2, the FPS is calculated in an exclusive code component that receives the number of frames that the video has and the total execution time in seconds (which is calculated by the "Chronometer" component).

To evaluate the general system performance and its tracking capability, we will use the measure known as F1-Score [89], also known as the Sørensen-Dice Coefficient, a variation of the F-Score metric that is defined by the following equation (Equation 4.6):

$$\mathrm{F} = (1 + \beta^2) \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{(\beta^2 \times \mathrm{Precision}) + \mathrm{Recall}} \tag{4.6}$$

where β is a factor that determines the weight that Precision and Recall will have in the result. The F result is a number that goes from 0 to 1 and represents the final success percentage. F1-Score is a F-Score variant where β = 1, which gives equal weight to both Precision and Recall resulting in a balanced result where both metrics have the same weight. When we substitute the β value in the original F-Score equation we get the next F1-Score equation (Equation 4.7):

$$\mathrm{F1} = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{4.7}$$

Precision and Recall are obtained through the following equations:

$$\mathrm{Precision} = \frac{TP}{TP + FP} \tag{4.8}$$

$$\mathrm{Recall} = \frac{TP}{TP + FN} \tag{4.9}$$

where TP = True Positives, FP = False Positives and FN = False Negatives, and these are obtained through the IoU (Intersection Over Union) measure. IoU is a metric that allows that tells us how much the truth and the ground truth have coincided (Figure 4.22). Like F-Score, IoU is a value that can go from 0 to 1 determining the correlation level of truth and ground truth, being a 1 a complete coincidence and 0 the opposite (the two bounding boxes don't touch each other at all).

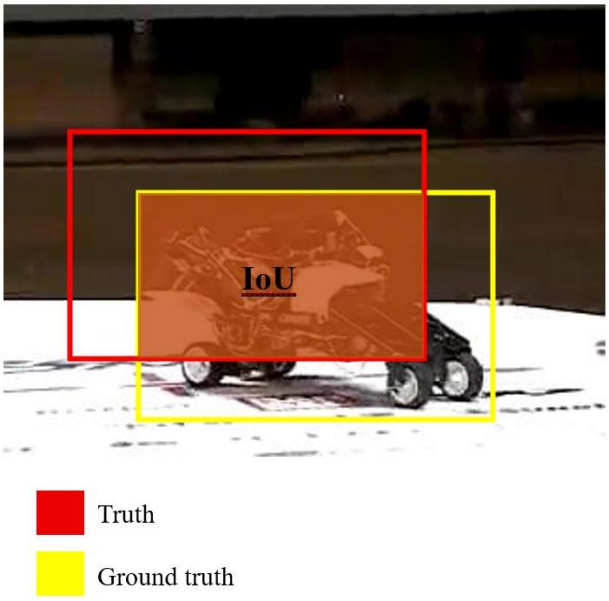

Figure 4.22. IoU is based on the intersection area between the truth and the ground truth

IoU is obtained though the next equations (Equations 4.8 – 4.12):

$$\mathrm{n} = \max\left(0, \min(x_T + h_T, x_{GT} + h_{GT}) - \max(x_T, x_{GT})\right) \tag{4.8}$$

$$\mathrm{m} = \max\left(0, \min(y_T + w_T, y_{GT} + w_{GT}) - \max(y_T, y_{GT})\right) \tag{4.9}$$

$$\mathrm{I} = \mathrm{n} * \mathrm{m} \tag{4.10}$$

$$\mathrm{U} = [(h_T * w_T) + (h_{GT} * w_{GT})] - \mathrm{I} \tag{4.11}$$

$$\mathrm{IoU} = \frac{I}{U} \tag{4.12}$$

where x_T and y_T are the coordinates of the truth, x_{GT} and y_{GT} are the coordinates of the ground truth, h_T and w_T are the height and width of the truth, and h_{GT} and w_{GT} are the height and width of the ground truth.

To calculate the true positives, false positives and false negatives, we use the PASCAL criterion [17] which establishes that if IoU >= 0.5 then it is a true positive, if IoU < 0.5 it's a false positive, and if the presence of ground truth confirms the existence of the object in the frame but the system says there is no object, then it is a false negative. It is worth mentioning that right now, this implementation assumes that the object is in all the

frames and for now there is not a mechanism to detect when the object isn't in the frame, so to calculate Recall we consider false negatives = 0.

Chapter 5

Experiments and results

5.1 Testing of parallel ZNCC versions

During the development of the system, the ZNCC component was created before the HSA component was created. As has been established in previous chapters, ZNCC algorithm can be used as a tracker by itself, so it was tested first. For both the ZNCC and HSA code components, different versions were released as improvements and fixes were added. The following table describes the main versions developed for this ZNCC implementation (Table 5.1):

Version	Description (improvements)
V1	New version made in Python of the original parallel implementation with no improvements over previous code.
V2	Implemented random default pixel percentage instead of the previous 100%.
V2.1	Calculation of $\bar{t}$ value of ZNCC equation (Equation 3.1) outside of parallel kernel (now only done once per frame, instead of in each kernel).
V2.2	Removal of unnecessary memory cleanup.
V2.3	Search area reduction: perimeter two times the size of the object around the last position.
V2.3.1	Search area reduction: predetermined percentage relative to frame size.

V2.3.1.1	Search area reduction: predetermined percentage relative to frame size. This improvement has been implemented to recover the search percentage lost in the previous version.
V2.3.2	Search area reduction: different predetermined percentages on the 4 sides of the object.
V2.3.2.1	Search area reduction: different random percentages on all four sides of the object. Experimental non-functional version.
V2.3.2.2	Search area reduction: different predetermined percentage on the four sides of the object. Percentage relative to the 3 times object's size radius.
V2.3.3	Blinds method.
V2.4	Random coordinates with search area reduction and OpenCL unique context improvement. Adapted version to work with HSA algorithm.
V2.5	Auto-adaptable search area, implementation of the formula to calculate optimal search radius (Equation 4.1). Adapted version to work with HSA algorithm.
V2.5.1	Auto-adaptable search area, implementation of the formula to calculate optimal search radius. Adapted to work alone without HSA algorithm.
V2.6	Implementation of the bubble compass (currently disabled). Adapted to work with HSA algorithm. **Definitive version.**

Table 5.1. Summary of ZNCC versions

The list above does not fully describe all the ZNCC versions developed, since (as mentioned in the previous chapter) the idea of the multi-kernel improvement was considered during development generating several versions that were eventually discarded

along with the improvement itself. The list above only details the relevant versions to understanding the evolution of the ZNCC implementation up to the current final version of its code component which is version 2.6.

As can be seen in the list of ZNCC versions, some early versions were not adapted to work coupled to the HSA components. These versions are those that were tested as individual trackers by themselves without any intervention of the HSA algorithm and their results led to the selection of the final version that would later be used as the fitness function of the HSA algorithm. As was mentioned in the previous chapter, all the versions of the system were tested on the computer whose specifications can be seen in Table 4.1. The F1-Score and speed results of the parallel single ZNCC versions processing the whole ALOV300++ dataset can be seen below (Table 5.2, and Figures 5.1 and 5.2):

Version	F1-Score	Speed
V1	55%	5.84 FPS
V2	48%	6.14 FPS
V2.1	49%	5.25 FPS
V2.2	49%	5.53 FPS
V2.3	50%	8.37 FPS
V2.3.1	52%	8.86 FPS
V2.3.1.1	51%	8.71 FPS
V2.3.2.1	45%	8.86 FPS
V2.3.2.2	54%	12.74 FPS
V2.3.3	52%	12.60 FPS
V2.5.1	47%	14.55 FPS

Table 5.2. F1-Score and speed of the single ZNCC parallel versions

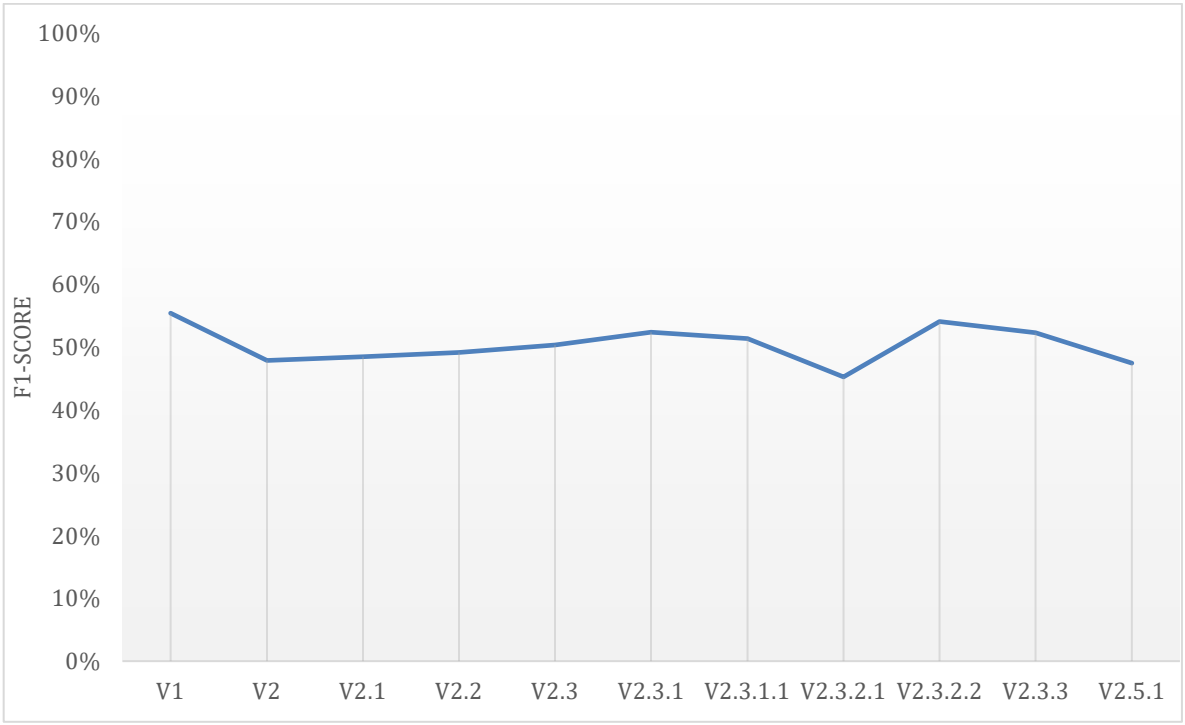

Figure 5.1. F1-Score graph of the single ZNCC parallel versions

Figure 5.2. Speed graph of the single ZNCC parallel versions

It is worth mentioning that some versions were tested more than once because they were tested under different parameter variations, especially the random bee population reduction and the search area reduction. The versions that implement these improvements were tested using 20% of bees (with respect to 100% of the total pixels of the frame) and a reduction of 75% of the search area (with respect to 100% of the total frame area).

As can be seen in the graphs for both F1-Score and speed (Figures 5.1 and 5.2), F1-Score remains stable with slight variations as the versions progress, while the speed increases progressively. Since it was the primary version without any improvement, V1 got a considerable good F1-Score (55%) but it also had a slow speed (5.84 FPS). In V2, the increase in speed is notorious (6.14 FPS) considering the drastic reduction in pixel area to analyze and there's also a decrease in F1-Score (48%) since this is a less exhaustive version but it is not a notable decrease which tell us that the reduction of pixels to process doesn't affect the F1-Score in a big way. V2.1 and V2.2 showed a progressive increase in F1-Score and a similar speed to V1 and V2 until V2.3 which increased the speed (8.37 FPS) thanks to the search area reduction improvement, this speed tendency was kept for the next 3 versions (V2.3.1, V2.3.1.1 and V2.3.2.1). As was expected, V2.3.2.1 showed a decrease in F1-Score since it is the version where the random search area change was implemented only for experimental purposes. V2.3.2.2 represented another increase in speed (12.74 FPS) since here the search area reduction began to be relative to 3 times the size of the object instead of relative to the total frame size. V2.3.3 didn't represent a big increase in both metrics. V2.5.1 showed a considerable increase in speed (14.55 FPS) but also had a decrease in F1-Score (47%) due to the recent improvements that were implemented with HSA in mind. It is worth mentioning that this version was created in order to test the test these improvements without the HSA intervention, that's why it is the same as V2.5.

5.2 Testing of parallel system

Like the ZNCC component, several versions of the HSA component emerged during its development as new improvements and corrections were integrated. These versions are described below (Table 5.3):

Version	Description (improvements)
V1	Basic Python implementation with no significant improvements over previous code.
V2	Implementation of parallel harvesting using Multiprocessing library (currently discarded). Version not tested.
V2.1	Implementation of parallel harvesting using Joblib library.
V2.2	Single ZNCC: instead of creating different ZNCC objects for each harvesting population, just one ZNCC object is created to process all harvester bees.
V2.2.1	Optimization of the harvesting parallel part with Joblib.
V2.3	Implementation of the optimal recruitment phase using exponential distribution (Equation 4.2).
V2.4	Adaptative population size based on search radius (Table 4.2).
V3	The exploration phase is executed only when it is needed.
V3.1	Re-assignation of rejected explorer bees as harvester bees.
V3.1.1	Discarded frame re-size improvement. Version not tested.

V3.1.2	Checkpoint frame improvement. **Pre-definitive version.**
V3.1.3	Acceleration improvement that reduces population after reaching a certain number of frames with a 0% IoU. **Definitive version.**

Table 5.3. Summary of HSA versions

The next tables and graphs show the F1-Score and speed results of these HSA versions (Table 5.4, and Figures 5.3 and 5.4):

Version	F1-Score	Speed
V1	37%	12.98 FPS
V2.1	35%	2.55 FPS
V2.2	38%	8.93 FPS
V2.2.1	10%	7.01 FPS
V2.3	33%	6.63 FPS
V2.4	48%	4.7 FPS
V3	47%	9.27 FPS
V3.1	47%	9.26 FPS
V3.1.2	48%	9.31 FPS
V3.1.3	54%	11.02 FPS

Table 5.4. F1-Score and speed of the HSA system versions

Figure 5.3. F1-Score graph of the HSA system versions

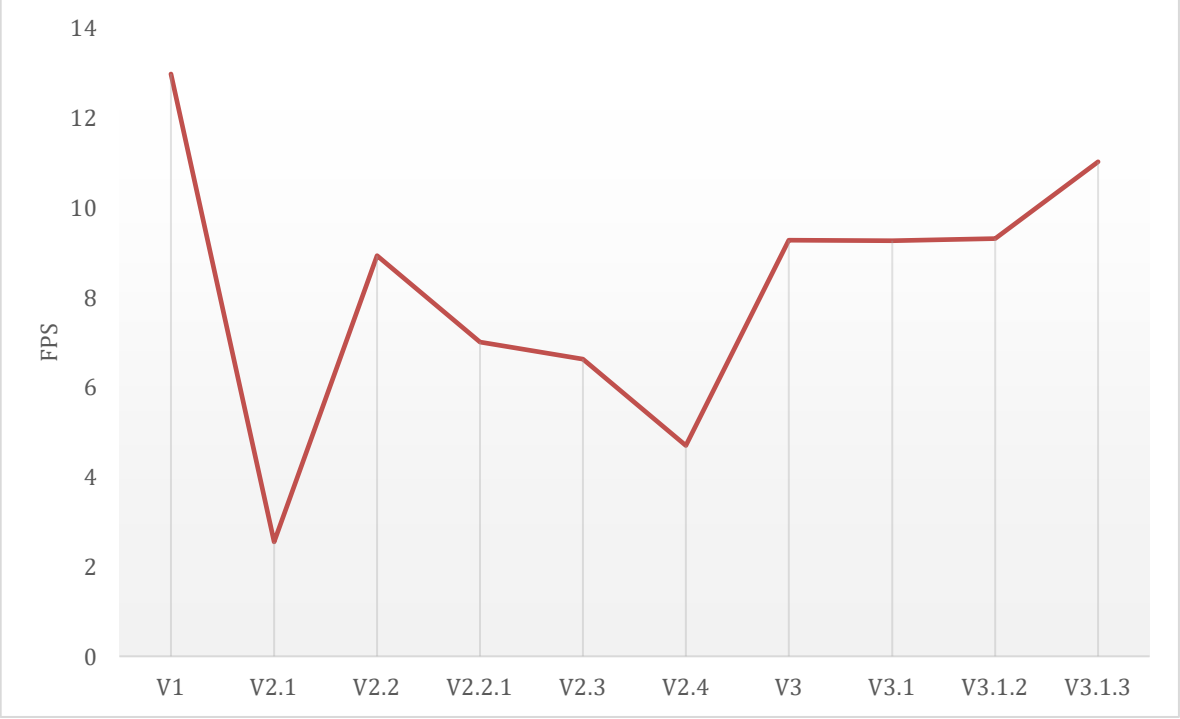

Figure 5.4. Speed graph of the HSA system versions

As can be seen, the first HSA versions got considerably low F1-Score, especially V2.2.1 which only got a 10% F1-Score because of the Joblib section's lack of fine-tuning at this point. But after these first versions, F1-Score in the following versions considerably increased until V3.1.3 (54%). In terms of speed, V1 starts with a considerably good value (12.98 FPS) to drastically decrease in V2.1 (2.55 FPS) due to the previously mentioned Joblib improvement which was not properly implemented yet. Then speed increases through the next versions until V2.4, which got a considerably low speed (4.7 FPS) due to the adaptative population improvement that was not correctly implemented in this version. Then, the speed increases thanks to the improvements until V3.1.3 (11.02 FPS). V3.1.3 was established as the definitive version of the system because it had the highest F1-Score, but before V3.1.2 was considered the definitive one until the acceleration improvement came up and V3.1.3 emerged.

More exhaustive tests were made on these two versions to calibrate some of the parameters, especially on V3.1.3. Also, each different test was performed 3 times, and the results that will be shown about these tests are the average result of the 3 tests. In version 3.1.2, the only tests that were performed were moving the IoU threshold that is used to save the current template as the new checkpoint frame. The following table (Table 5.5) shows the F1-Score results of V3.1.2 under different checkpoint frame update IoU thresholds:

IoU threshold	F1-Score	Speed
0.0	51%	9.52 FPS
0.4	47%	9.31 FPS
0.6	50%	9.38 FPS
0.8	58%	9.29 FPS
0.9	55%	9.06 FPS
0.95	54%	9.19 FPS

Table 5.5. F1-Score and speed results using different IoU threshold on checkpoint frame on version 3.1.2

As can be seen, the checkpoint frame improvement is more effective when we consider IoU <= 0.9 to update the checkpoint frame this is because we ensure that the best checkpoint template is saved as checkpoint frame making it resilient to small changes like

rotations or light changes, a resilience it wouldn't have at a higher threshold (like 0.95), so 0.9 is the threshold value that stayed in the next tests on version 3.1.3.

In version 3.1.3, the first tests were related to the acceleration improvement. As is explained in section 4.4.1, this improvement reduces drastically the bee population after a certain number of frames where IoU = 0.0 at the beginning of the video. To calibrate the number of consecutive zeros ("zero tolerance") that the system should tolerate at the beginning of the video before reducing the population, we tested three different values: 3, 5 and 7. It is worth noting that (as was explained in previous chapters) due to the nature of the ALOV300++ dataset, the IoU evaluation is not done every single frame and there is always a certain number of frames between one frame that is evaluated and the next one, so this zero tolerance only consider the frames where IoU is evaluated. Following table (Table 5.6) shows the F1-Score and speed results of the zero tolerance tests:

Zero tolerance	F1-Score	Speed
3	51%	11.22 FPS
5	53%	10.98 FPS
7	54%	11.02 FPS

Table 5.6. F1-Score and speed results using different zero tolerance values

Although it was not the fastest result, the 7 zeros tolerance was the configuration with the highest F1-Score allowing the system to give the video a chance to "recover" or find the object without lose the purpose of this improvement (something that could happen with a bigger zero tolerance), so it was the one that stayed. Another parameter that was tested is the size of the search radius, specifically the α parameter (Equation 4.1) that controls the search area size in relation to the object's size as it is explained in section 4.3. The following tests consisted of reducing and increasing the search radius. While the α value up to that point was 1.5, for the tests it was set at 0.5, 1.0, 2.0, and 2.5. The results of these tests can be seen below (including the previous results using α = 1.5) (Table 5.7, and Figures 5.5 and 5.6):

α	F1-Score	Speed
0.5	51%	14.14 FPS
1.0	53%	12.23 FPS
1.5	54%	11.02 FPS
2.0	56%	9.75 FPS
2.5	57%	8.75 FPS

Table 5.7. F1-Score and speed results using different α values

As can be seen, as we increase the α value, F1-Score increases because there's a more exhaustive search and a bigger chance to find the object, but also the speed decreases because there's a bigger frame area to analyze so in this case we opted to keep = 1.5 for the next tests that consisted of increasing and decreasing the bee population. It is worth mentioning that both, the α value for search area tests, and the population size tests, were tests focused on find the maximum potential of the system in both metrics (F1-Score and speed) although this maximum value will not be reached with the same configuration as we can see with the adjustment of the α parameter which increases F1-Score at the expense of speed and vice versa.

Something similar happened on the population size tests. As was explained in section 4.4.1, the bee population size is assigned according to the size of the search radius (Table 4.2) and just like with the search area, as we increase the bee population, F1-Score increases but speed decreases. To these final tests, we increased and divided the population 2 and 4 times in each case, the results can be seen below (Table 5.8):

		F1-Score	Speed
Decrement	x4	34%	12.59 FPS
	x2	45%	11.54 FPS
Increment	x2	59%	8.22 FPS
	x4	62%	6.71 FPS

Table 5.8. F1-Score and speed results by increasing and decreasing bee population

As we said, these tests proved that for now, the system is unable to achieve the proposed goal of a 62% F1-Score and a 15 FPS speed at the same time using a single configuration. Taking this into account, we consider that the most optimal configuration

with the most balanced results in both F1-Score and FPS is the one presented above (Table 5.7) with the search area parameter $\alpha = 2.5$. Following table (Table 5.9) shows F1-Score and FPS results from each category of the ALOV dataset on this configuration:

Category	F1-Score	FPS
01-Light	55%	11.41 FPS
02-SurfaceCover	60%	8.02 FPS
03-Specularity	67%	4.7 FPS
04-Transparency	69%	9.83 FPS
05-Shape	48%	5.58 FPS
06-MotionSmoothness	47%	9.37 FPS
07-MotionCoherence	45%	13.03 FPS
08-Clutter	72%	9.37 FPS
09-Confusion	59%	9.41 FPS
10-LowContrast	70%	7.39 FPS
11-Occlusion	57%	7.6 FPS
12-MovingCamera	64%	7.73 FPS
13-ZoomingCamera	44%	0.03 FPS
14-LongDuration	25%	12.02 FPS
Dataset's average	**57%**	**8.76 FPS**

Table 5.9. F1-Score and speed results from each category on the most optimal system configuration

On the other hand, these tests showed the maximum potential that the ALOV300++ dataset videos could achieve in both F1-Score and speed under certain parameters of the current system's implementation. The next table (Table 5.9) shows the F1-Score and speed results of each category of the ALOV dataset and the whole dataset itself considering the maximum result got in each video in each test:

Category	F1-Score	FPS
01-Light	70%	17.62 FPS
02-SurfaceCover	70%	11.81 FPS
03-Specularity	72%	9.73 FPS
04-Transparency	81%	15.08 FPS
05-Shape	58%	11.15 FPS
06-MotionSmoothness	59%	15.06 FPS
07-MotionCoherence	68%	16.57 FPS
08-Clutter	81%	16.97 FPS
09-Confusion	74%	15.27 FPS
10-LowContrast	83%	13.39 FPS
11-Occlusion	67%	13.16 FPS
12-MovingCamera	76%	13.97 FPS
13-ZoomingCamera	55%	15.04 FPS
14-LongDuration	41%	16.63 FPS
Dataset's average	**68%**	**14.39 FPS**

Table 5.10. Maximum F1-Score and speed results from each category

As can be seen, the best results from the whole dataset surpassed the target of 62% F1-Score, achieving 68%. In speed, system didn't achieve the goal of 15 FPS but came close with 14.39 FPS. Next graphs (Figures 5.5, 5.6, 5.7 and 5.8) show the distribution of all the ALOV300++ dataset videos among the different F1-Score and speed scores:

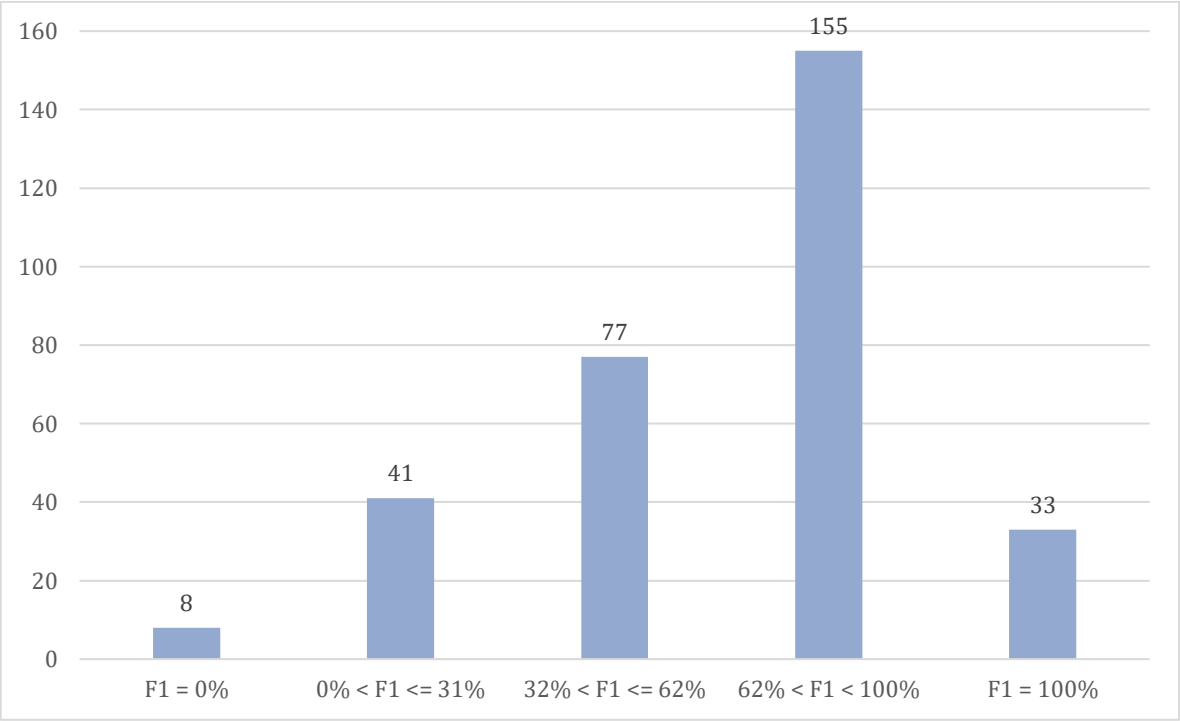

Figure 5.5. Videos that achieve certain F1-Score (F1) range

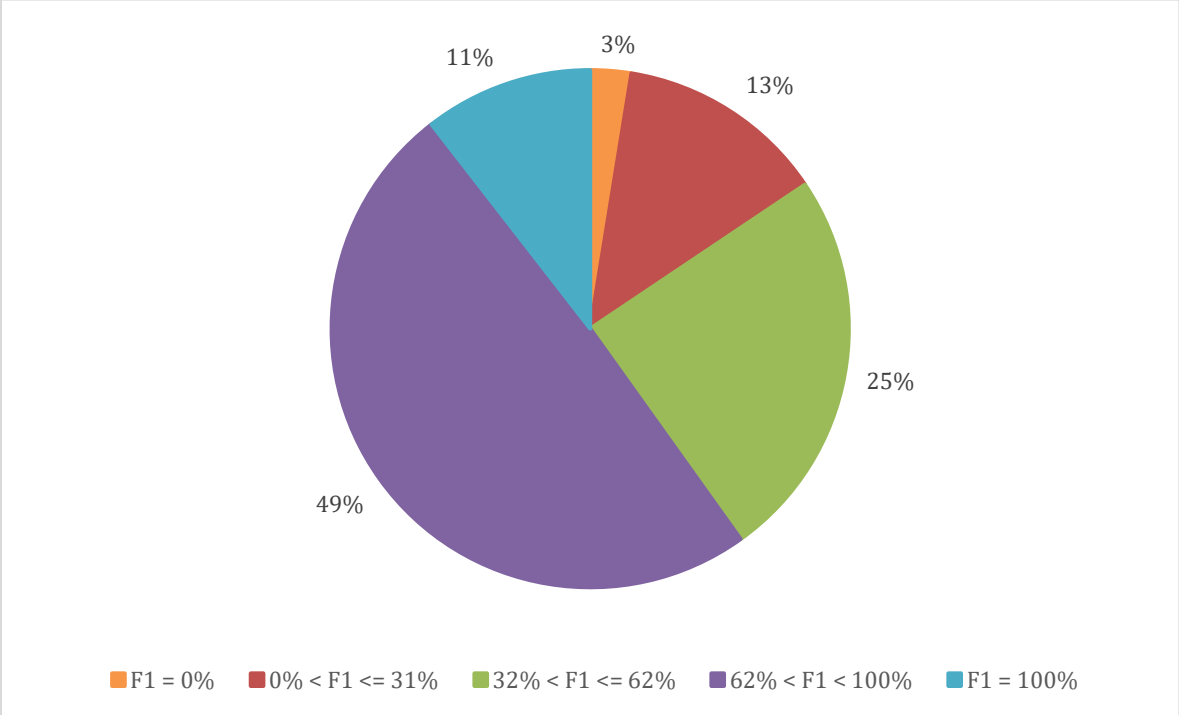

Figure 5.6. Percentage of videos that achieve certain F1-Score range

As can be seen, even if the F1-Score results are not perfect, they are promising since the F1 range where a big number of videos fell was the one that goes from more than 62% to less than 100%, which implies that more than half of the videos overpassed the goal of a 62% F1-Score (this includes the 33 videos that achieved a 100%). Also, the smallest group of videos were those that got a 0% F1-Score.

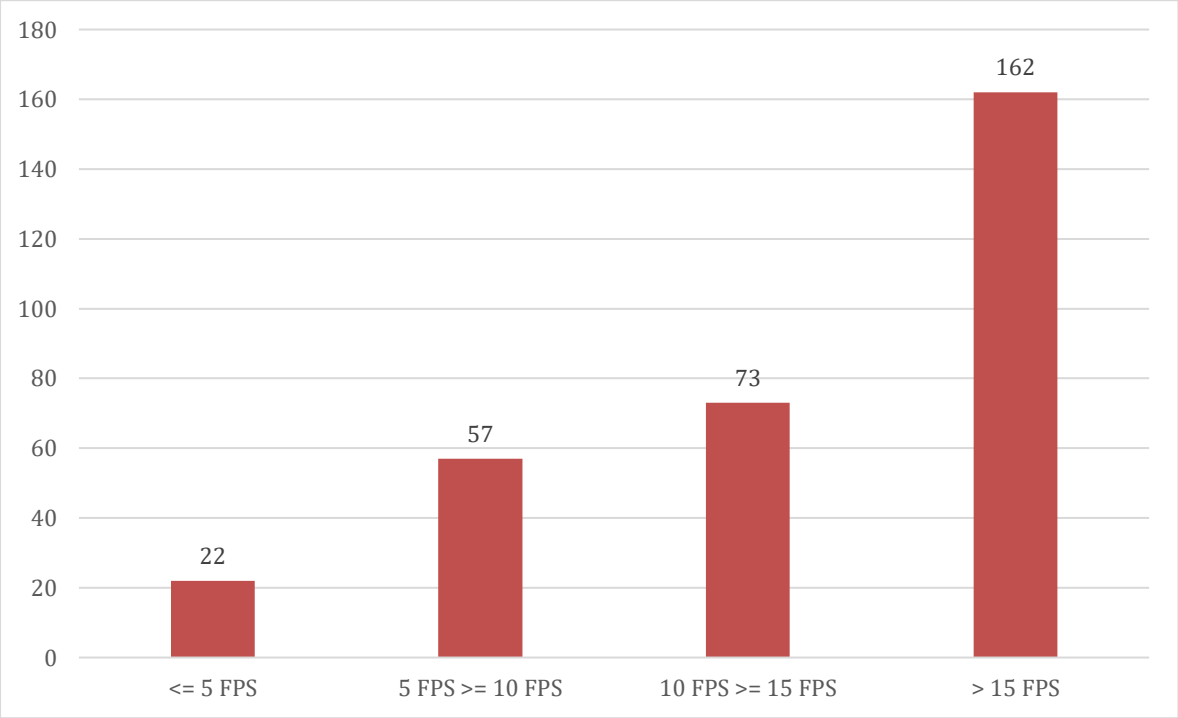

Figure 5.7. Videos that achieve certain FPS range

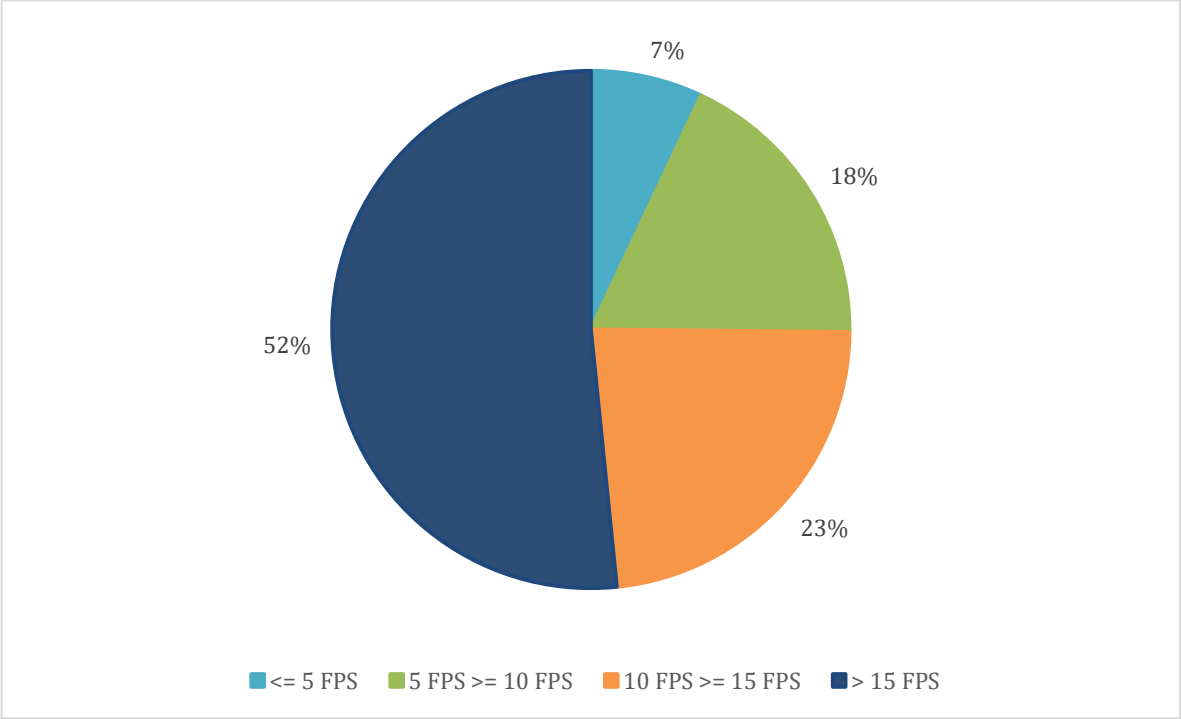

Figure 5.8. Percentage of videos that achieve certain FPS range

On the speed side, more than half of the videos exceeded the 15 FPS goal, which is promising but not enough for reach 15 FPS on the whole dataset. Fortunately, less than 10% of the videos got less than 5 FPS.

5.3 Comparison to recent proposals

Following the previous results, the next table (Table 5.10) makes an F-Score comparison (and category comparison based on the classification described on Chapter 3 and proposed by Smeulders et al. [11]) between this new implementation, the previous one [15] and some other proposals on ALOV300++ dataset [11] [17] [90] [91]:

Proposal	F-Score	Online category
DeepSRDCF	79%	Discriminative Classification with Constraints

Spatially Regularized Discriminative Correlation Filter (SRDCF)	78%	Discriminative Classification with Constraints
Scale Adaptive with Multiple Features Tracker (SAMF)	77%	Discriminative Classification with Constraints
DeepDCF	76%	Discriminative Classification with Constraints
Discriminative Scale Space Tracker (DSST)	75%	Discriminative Classification with Constraints
PRO Tracker	74%	Discriminative Classification with Constraints
Channel and Spatial Reliability Tracker (CSRT)	70%	Discriminative Classification with Constraints
Target-Guided Regression for Tracking (TGRP)	69%	Discriminative Classification with Constraints
New implementation (HSA-Py)	68%	Matching with Extended Appearance Model
Struck (STR)	66%	Discriminative Classification with Constraints
Multiple Experts using Entropy Minimization (MEEM)	65%	Discriminative Classification
Foreground-Background Tracker (FBT)	64%	Discriminative Classification
Multiple Instance Learning Tracker (MIL)	64%	Discriminative Classification
Visual Tracking via Spare Representation (VTS)	62%	Matching with Extended Appearance Model
Tracking by Sampling Trackers (TST)	62%	Matching with Extended Appearance Model

Tracking-Learning-Detection (TLD)	61%	Discriminative Classification with Constraints
SiamMask	61%	Discriminative Classification
L1 Tracker with Occlusion Detection (L1O)	60%	Matching with Extended Appearance Model
Normalized Cross-Correlation (NCC)	57%	Matching
Multiple Instance Learning Tracker (MIL)	57%	Discriminative Classification
L1-minimization Tracker (L1T)	57%	Matching with Constraints
Multiple Instance Tracker (MIT)	57%	Discriminative Classification
Incremental Visual Tracking (IVT)	55%	Matching with Extended Appearance Model
Kernelized Correlation Filter (KCF)	55%	Discriminative Classification with Constraints
Fragments-based Robust Tracker (FRT)	52%	Matching with Extended Appearance Model
Locally Orderless Tracking (LOT)	52%	Matching
Hough-Based Tracking (HBT)	49%	Discriminative Classification
Lucas-Kanade Tracker (KLT)	47%	Matching
Super Pixel Tracking (SPT)	47%	Discriminative Classification
Kalman Appearance Tracker (KAT)	43%	Matching
Tracking on the Affine Group (TAG)	38%	Matching with Extended Appearance Model
Mean Shift Tracker (MST)	36%	Matching
Adaptive Coupled-layer Tracking (ACT)	27%	Matching with Constraints

Previous implementation (HSA-C)	24%	Matching
Tracking by Monte Carlo sampling (TMC)	15%	Matching with Constraints

Table 5.11. F1-Score and classification of different trackers including the one proposed in this research

As can be seen from the table, our proposal achieved a relatively good performance outperforming most of the trackers shown above (including the previous implementation of the system), but it still does not reach the best ones. This can also be seen in the following graph (Figure 5.9) that compares the survival curve of this proposal with some others in the table above:

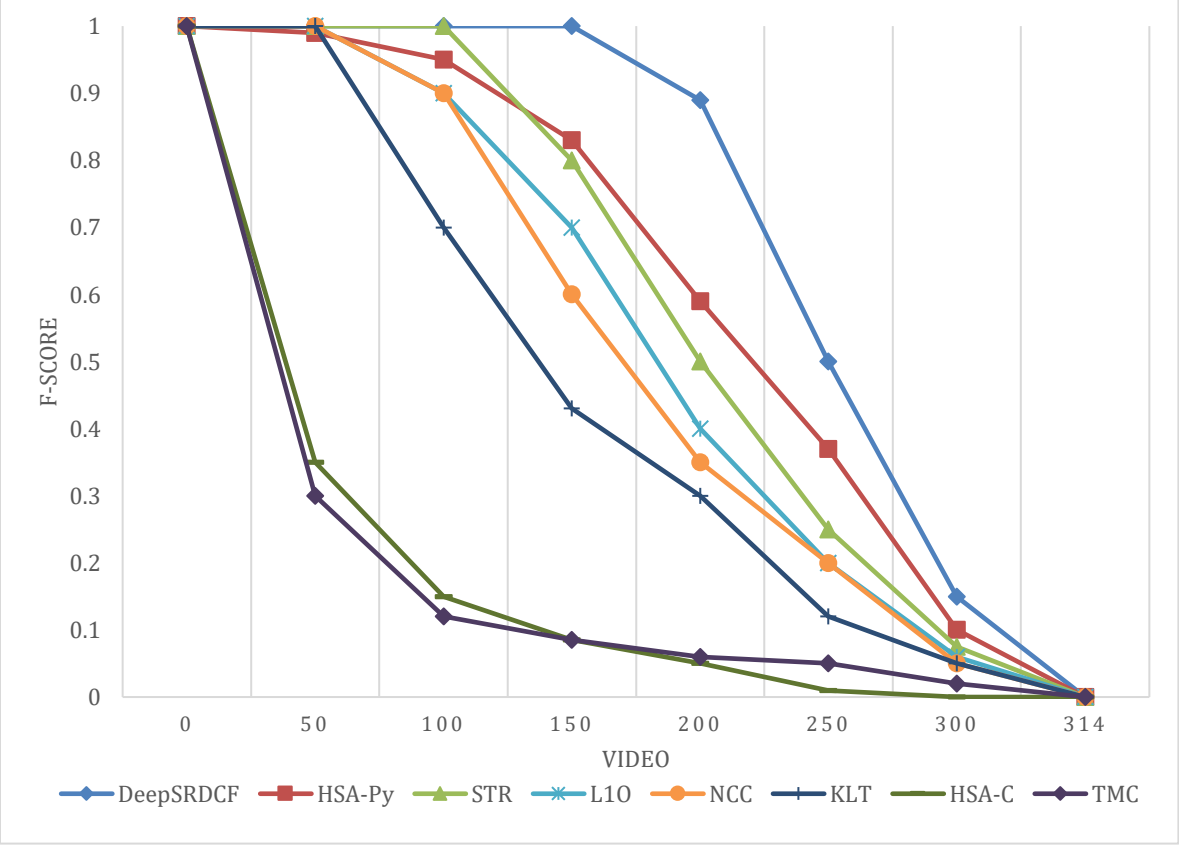

Figure 5.9. Survival curves of the actual proposal vs. other proposals

The survival graph seen above shows a performance comparison between our implementation and some others on the ALOV300++ dataset. On the graph, our implementation stands out to be one of the more stable and closer to a linear distribution, which is not the ideal survival curve since it does not stay in the upper limit for so long, but it also doesn't have an abrupt fall like other curves. Next graph (Figure 5.10) makes an F-Score comparison between our implementation, the previous implementation and one from each online category [11]:

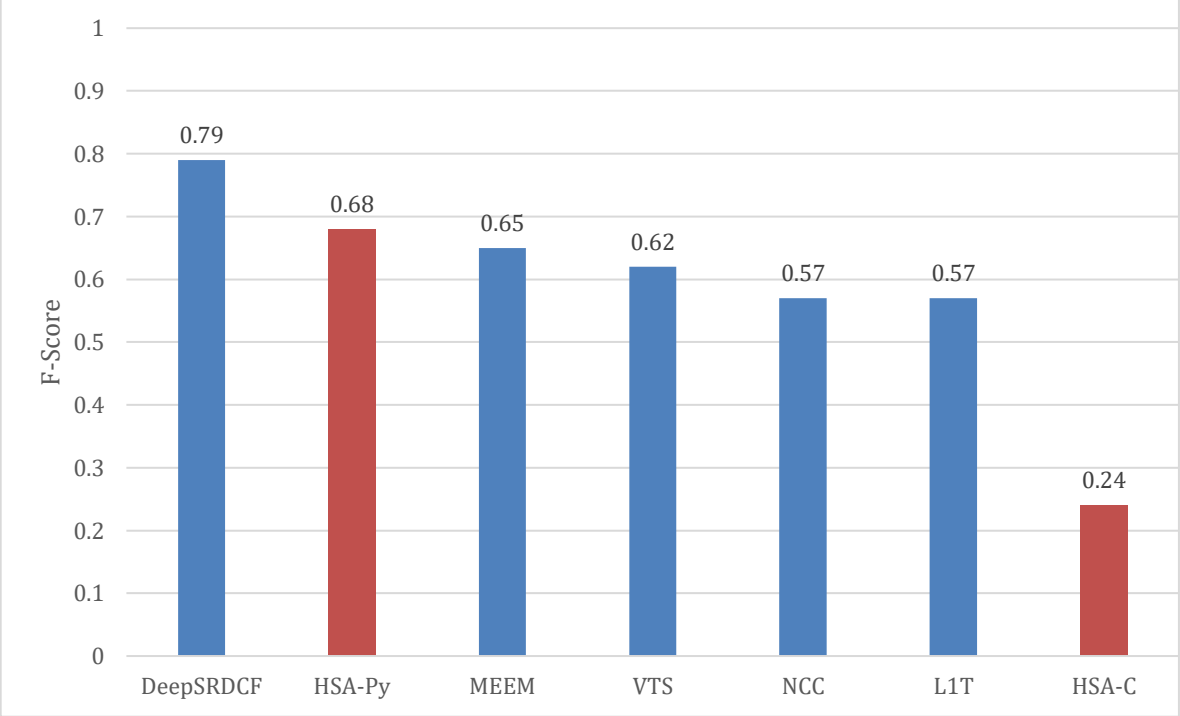

Figure 5.10. F-Score comparison between our proposal and others

Chapter 6

Conclusions and future work

As was seen in the previous chapter, this system proposal achieved satisfactory results. The following list describes the extent to which the specific objectives of this thesis (described in section 1.4) were achieved:

1. The previous work related to GPU-based video tracking system programmed in C using HSA and ZNCC was successfully tested and analysis of its areas of opportunity was done. A description of the PC where this previous implementation was tested can be found in Chapter 4 (Table 4.1) and a comparison of these results and a comparison between these results and those of the current proposal can be seen in Chapter 5.
2. A successful design for this new implementation was made and implemented for a CPU-GPU architecture taking advantage of Python's multi-paradigm nature and its optimized and specialized libraries although Python is an interpreted language, not a compiled one, which could make it slower than other languages like C and C++.
3. Even if one single configuration of the system can achieve both, F1-Score = 62% and speed = 15 FPS, in one run, the maximum results that were gotten prove the potential of the system with F1-Score = 68% and speed = 14.39 FPS. But the system has already been able to achieve 62% in one run with one configuration which confirms that this implementation surpasses the previous one [15] and is on par with other recognized trackers such as Struck and SiamMask.
4. As has been said, maximum results got an average of 14.39 FPS for the system, which is close to the goal of 15 FPS and represents a considerable improvement

over the previous implementation thanks to the improvements described in previous chapters which perform a more optimized and intelligent frame-by-frame search for the target, greatly reducing the noise and redundancies present in the exhaustive search that was made on the previous implementation. Using more powerful hardware, the system could be capable of surpassing the 15 FPS goal.

5. The system was fully tested using the ALOV300++ dataset. All the tests are described in Chapter 5.

The results per category shown in the previous chapter (Table 5.8) demonstrate that the system has a good performance in almost every category of the ALOV300++ dataset, being 14-LongDuration the only category with a relatively low F1-Score (41%), this tells us that the system may have difficulty processing long videos. On the other hand, the category with the higher F1-Score score was 10-LowContrast (83%), followed by 04-Transparency and 08-Clutter (81%, both categories), so the system could be useful for video surveillance, especially in busy environments or with low-contrast conditions. It could also be useful for other applications with these characteristics such as microscopy (analysis of cells, viruses, bacteria, etc...) or robotics for handling transparent objects (such as industrial robots). In terms of speed, the system maintains an average of around 14 FPS across all categories, with the fastest being 01-Light (17.62 FPS) followed by 08-Clutter (16.97 FPS), reaffirming the system's strength in cluttered environments. On the other hand, the category with the lowest average speed was 03-Specularity (9.73 FPS), which nevertheless achieved a reasonable F1-Score (72%). These speed values fit very well with several of the applications mentioned above, especially video surveillance.

But even if this implementation showed promising results, it didn't achieve the expected goals. As was said, one configuration of the system achieved a 62% F1-Score, but at a high cost in speed, so there's still work to be done to keep this results without sacrificing the system's response time. On one hand, a more thorough analysis of the implemented improvements as well as the parameters of the algorithm in general is recommended to accurately determine their real impact on system performance, especially those implemented in versions that exhibited certain anomalies or sudden changes in F1-

Score or speed during testing (see Chapter 5). Some of the improvements and details that must be checked are:

- Hardware selection: as was established in section 4.1, the system was tested on one single computer, so it is recommended to run tests on different computers with different hardware components (especially different CPU and GPU models) to compare how hardware affects system's performance.
- Software selection: as has been said, the system was programmed in Python, which has the disadvantage of being an interpretated language which could make the implementation slowly than if it were made on a compiled language, so One opportunity for improvement could be to create a new implementation of the system in a compiled language that in turn allows maintaining the component-based and flexible nature of the code, as is the case with C++, which, although it does not have several of the specialized libraries of Python, could represent a considerable improvement in speed. Another option would be to keep Python language but test some of its other libraries for specialized tasks (such as image processing or evolutionary computing).
- Optimization of code design: even if the design of the system successfully provides a satisfactory flexibility, future revisions of the code could help make the design more adaptable to take more advantage of the multi-paradigm nature of Python.
- Optimization of parallel implementation: future redesigns could also lead to an optimization of the parallel programming part could also be carried out to better distribute tasks between execution threads on CPU and GPU.
- Review of the automatic search area: it is recommended to do a meticulous review on the implementation of the formula that establishes the search radius for each video (Equation 4.1), since during the testing phase it was observed that its implementation with different parameters affects each video differently.
- Review of the blinds method: it is also recommended to conduct a thorough review of both the positive and negative effects of the blinds method on the system.

- Optimization of HSA phases: it is also recommended to continue exploring ways to optimize and improve the 3 basic phases of HSA, especially reviewing and continuing to experiment with different combinations of evolutionary operators, evolutionary strategies and parameters in search of what could be a better configuration.
- Analysis of OpenCL memory cleanup and context: as was mentioned in section 4.4.3, OpenCL memory cleanup was removed just like the necessity of creating a different OpenCL execution context every single frame, this to optimize execution time. Even if these improvements prove to accelerate the system, it is recommended to keep analyzing the impact of both the presence and the absence of these two OpenCL elements in all the aspects of the system.
- Review of evaluation metrics: although F1-Score (along IoU, Precision and Recall) and FPS are widely used evaluation metrics, it is recommended to consider other metrics such as GIoU in order to evaluate the system more comprehensively.

On the other hand, some of the improvements discarded (like those described in section 4.4.3) could have potential if they are properly implemented. Some of these improvements are:

- Multi-kernel: although the multi-kernel improvement demonstrated a high negative redundancy, some of the ideas from this improvement could be redirected and partially implemented in the future to truly optimize the processing of the ZNCC algorithm.
- Bubble compass: this was another promising improvement that in theory sounds quite beneficial to the system, but due to lack of time it could not be implemented correctly.
- Resilience to rotation: although the system currently possesses some resilience to target rotation thanks to the constant updating of the object's template, a formal mechanism capable of handling the rotation challenge appropriately has not yet been implemented.

- Resilience to size change: in the case of size change, this is currently a major weakness of the system since there is no mechanism that considers the size change of the object or even a modification on bounding box's dimensions, so it is an option that must be considered in future work.
- Color processing: another weakness of the system is that, currently, the algorithm works converting the images into grayscale. The possibility of handling color images was discussed during development, but this was not implemented due to time constraints and the complexity of the problem. However, if image processing can be achieved while maintaining the original RGB tones, it could represent a considerable improvement.

Also, it is worth mentioning that due to component-based nature of the system, it could then be used as a platform to test other VOT algorithms, even using other forms of object representation (like those described in section 3.1) or evaluation metrics (like those described in section 3.4).

Bibliography

[1] Alper Yilmaz, Omar Javed & Mubarak Shah. (2006). Object tracking: A survey. *ACM: Digital Library*, 38(4), 45–58.

[2] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing & Junjie Yan. (2018). SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2019*, 4277-4286.

[3] Botao Ye, Hong Chang, Bingpeng Ma, Shiguang Shan & Xilin Chen. (2022). Joint Feature Learning and Relation Modeling for Tracking: A One-Stream Framework. *European Conference on Computer Vision 2022*, 341–357.

[4] Emilio Maggio & Andrea Cavallaro. (2011). Video Tracking: Theory and Practice. *Wiley Online Library*.

[5] Peize Sun, Jinkun Cao, Yi Jiang, Rufeng Zhang, Enze Xie, Zehuan Yuan, Changhu Wang & Ping Luo. (2021). TransTrack: Multiple Object Tracking with Transformer. *ArXiv, abs/2012.15460.*

[6] Weiming Hu, Qiang Wang, Li Zhang, Luca Bertinetto & Philip Torr. (2022). SiamMask: A Framework for Fast Online Object Tracking and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3), 3072–3089.

[7] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu & Xinggang Wang. (2022). ByteTrack: Multi-Object Tracking by Associating Every Detection Box. *European Conference on Computer Vision 2022*, 13682–13703.

[8] Yi Wu, Jongwoo Lim & Ming-Hsuan Yang. (2013). Online Object Tracking: A Benchmark. *Computer Vision and Pattern Recognition Conference 2013*, 2411–2418.

[9] Yutao Cui, Cheng Jiang, Limin Wang & Gangshan Wu. (2022). MixFormer: End-to-End Tracking with Iterative Mixed Attention. *IEEE Computer Vision and Pattern Recognition Conference (CVPR) 2022*, 13598–13608.

[10] Juan Alvaro de la Rosa-Ipiña, Carlos Soubervielle-Montalvo, Cesar Puente, Ruth Mariela Aguilar-Ponce, Luis Javier Ontañon García-Pimentel. (2025). Parallel Video Tracking System based on Honeybee Swarm Behavior and Evaluated on a Benchmark Dataset. *Research in Computing Science*, 153(10), 31–43.

[11] Arnold Smeulders, Dung Manh Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan & Mubarak Shah. (2014). Visual Tracking: An Experimental Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 36(7), 1442–1468.

[12] Wei Chen. (2022). Simulation of Multimedia Visual Image Motion Track Marking Based on Artificial Bee Colony Algorithm. *Mathematical Problems in Engineering*, 2022(10), 1–10.

[13] Hathiram Nenavath, Ravi Kumar Jatoth & Swagatam Das. (2018). A synergy of the sine-cosine algorithm and particle swarm optimizer for improved global optimization and object tracking. *Swarm and Evolutionary Computation*, 43, 1–30.

[14] Ming-Liang Gao, Jin Shen, Li-Ju Yin, Wei Liu, Guo-Feng Zou, Hai-Tao Li & Gui-Xia Fu. (2015). A novel visual tracking method using bat algorithm. *Neurocomputing*, 177, 612–619.

[15] Oscar Ernesto Perez-Cham. (2017). Parallelization of the Honeybee Search Algorithm for Video Tracking (Master thesis). *Universidad Autónoma de San Luis Potosí*.

[16] Oscar Ernesto Perez-Cham. (2021). Development of Heterogeneous Systems Based on the Honeybee Search Algorithm for Video Tracking (PhD thesis). *Universidad Autónoma de San Luis Potosí*.

[17] Oscar Ernesto Perez-Cham, Cesar Puente, Carlos Soubervielle-Montalvo, Gustavo Olague, Carlos Arturo Aguirre-Salado & Alberto Salvador Nuñez-Varela. (2020). Parallelization of the Honeybee Search Algorithm for Object Tracking. *Applied Sciences*, 10(6), 2122–2146.

[18] Carlos Soubervielle-Montalvo, Oscar Ernesto Perez-Cham, Cesar Puente, Emilio Jorge González-Galván, Gustavo Olague, Carlos A. Aguirre Salado, Juan C. Cuevas Tello & Luis Javier Ontañon Garcia-Pimentel. (2022). Design of a Low-Power Embedded System Based on a SoC-FPGA and the Honeybee Search Algorithm for Real-Time Video Tracking. *Sensors*, 22(3), 1280–1306.

[19] Víctor Alejandro Méndez-López, Carlos Soubervielle-Montalvo, Alberto Salvador Núñez-Varela, Oscar Ernesto Pérez-Cham & Emilio Jorge González-Galván. (2023). A survey on FPGA-based design methodologies for visual object tracking. *Abstraction & Application*, 42, 102–103.

[20] Mekhriddin Rakhimov, Jamshid Elov, Utkir Khamdamov, Shavkatjon Aminov & Shakhzod Javliev. (2021). Parallel Implementation of Real-Time Object Detection using OpenMP. *International Conference on Information Science and Communications Technologies (ICISCT) 2021*, 1–4.

[21] Longxiang Xu & Guosheng Wu (2024). Multi-Object Tracking with Grayscale Spatial-Temporal Features. *Applied Sciences*, 14(13), 5900 – 5913.

[22] Jiuhong Jiang, Yu Yin, Guangyu Zheng, Liang Ni & Zhenhao Hu. (2024). Parallel Optimization of Correlation Filter Tracking Based on CUDA. *ICITEE '23: Proceedings of the 6th International Conference on Information Technologies and Electrical Engineering*, 199–206.

[23] Ugur Taygan & Adnan Ozsoy. (2020). Performance analysis and GPU parallelisation of ECO object tracking algorithm. *New Trends and Issues Proceedings on Advances Pure and Applied Sciences*, 2020(12), 109–118.

[24] Sana Pavan Kumar Reddy, Jonnadula Harikiran & Bolem Sai Chandana. (2024). Deep CNN Based Multi Object Detection And Tracking In Video Frames With Mean Distributed Feature Set. *Procedia Computer Science*, 235, 723–734.

[25] Aleksandr Solovyev & Roman Solovyev. (2023). VOT Challenge: Computer Vision Competition. *Communications of the ACM*. https://cacm.acm.org/blogcacm/vot-challenge-computer-vision-competition/

[26] Amsterdam Library of Ordinary Videos for tracking (ALOV++). (s/f). *University of Central California - Center for Research in Computer Vision*. https://www.crcv.ucf.edu/research/data-sets/alov/

[27] Upasna Singh, Anjali Saini & Ravi Domala. (2020). Object Tracking in Videos Using CNN. *International Conference on Data Science, Machine Learning and Applications (ICDSMLA) 2019*, 601, 520–527.

[28] Wolfgang Banzhaf, Peter Nordin, Robert Keller & Frank Francone. (1998). Genetic Programming - An Introduction. *Morgan Kaufmann Publishers.*

[29] Marco Dorigo & Thomas Stützle. (2004). Ant Colony Optimization. *The MIT Press*.

[30] Xin-She Yang. (2010). A New Metaheuristic Bat-Inspired Algorithm. *Studies in Computational Intelligence, Springer, Berlin*, 284, 65–74.

[31] Cesar Puente. (2005). Reconstrucción tridimensional a partir de dos imágenes basada en el proceso de búsqueda de las abejas (Master thesis). *Centro de Investigación Científica y de Educación Superior de Ensenada.*

[32] David Edward Goldberg. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesle*.

[33] Agoston Endre Eiben & Jim Smith. (2015). Introduction to Evolutionary Computing. *Springer*.

[34] Thomas Baeck, David Fogel & Zbigniew Michalewicz. (2000). Evolutionary Computation 1: Basic Algorithms and Operators. *CRC Press*.

[35] Tobias Blicke & Lothar Thiele. (1996). A Comparison of Selection Schemes Used in Evolutionary Algorithms. *Evolutionary Computation*, 4(4), 361–394.

[36] Larry Eshelman & James David Schaffer. (1993). Real-Coded Genetic Algorithms and Interval-Schemata. *Foundations of Genetic Algorithms*, 2, 187–202.

[37] Adriana Cervantes, Efrén Mezura-Montes & Carlos Artemio Coello Coello. (2015). An empirical comparison of two crossover operators in real-coded genetic algorithms for

constrained numerical optimization problems. *IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC) 2014*, 1 – 5.

[38] Zbigniew Michalewicz. (1996). Genetic Algorithms + Data Structures = Evolution Programs. *Springer*.

[39] Hans-Paul Schwefel. (1995). Evolution and Optimum Seeking. *Wiley*.

[40] Victor Alejandro Méndez-López, Carlos Soubervielle-Montalvo & Emilio Jorge González-Galván. (2024). A Review of Design Methodologies and Evaluation Techniques for FPGA-Based Visual Object Tracking Systems. *International Journal of Combinatorial Optimization Problems and Informatics*, 15(5), 127–145.

[41] Meng Li, Zemin Cai, Chuliang Wei & Ye Yuan (2015). A Survey of Video Object Tracking. *International Journal of Control and Automation*, 8(9), 303–312.

[42] Kai Hanebeck & Uwe Hanebeck. (2001). Template matching using fast normalized cross correlation. *Proceedings of SPIE - The International Society for Optical Engineering,* 4387, 95–102.

[43] Simon Matthews & Iain Matthews. (2004). Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56, 221–255.

[44] Hieu Nguyen & Arnold Smeulders. (2004). Fast occluded object tracking by a robust appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8), 1099–1104.

[45] David Ross, Jongwoo Lim, Ruei-Sung Lin & Ming-Hsuan Yang. (2008). Incremental Learning for Robust Visual Tracking. *International Journal of Computer Vision*, 77(1–3), 125–141.

[46] Junghyun Kwon, Kyoung Mu Lee & Frank Chongwoo Park. (2009). Visual tracking via geometric particle filtering on the affine group with optimal importance functions. *IEEE Conference on Computer Vision and Pattern Recognition (CCVPR) 2009*, 991–998.

[47] Junseok Kwon & Kyoung Mu Lee. (2011). Tracking by Sampling Trackers. *IEEE International Conference on Computer Vision (ICCV) 2011*, 1195-1202.

[48] Junseok Kwon & Kyoung Mu Lee. (2009). Tracking of a Non-Rigid Object via Patch-based Dynamic Appearance Modeling and Adaptive Basin Hopping Monte Carlo Sampling. *IEEE Conference on Computer Vision and Pattern Recognition (CCVPR) 2009*, 1208-1215.

[49] Luka Čehovin, Matej Kristan & Alesš Leonardis. (2011). An adaptive coupled-layer visual model for robust visual tracking. *International Conference on Computer Vision (ICCV) 2011*, 1363–1370.

[50] Xue Mei & Haibin Ling. (2009). Robust Visual Tracking Using l1 Minimization. *IEEE 12th International Conference on Computer Vision (ICCV) 2009*, 1436–1443.

[51] Hieu Tat Nguyen & Arnold Smeulders. (2006). Robust track using foreground-background texture discrimination. *International Journal of Computer Vision*, 68(3), 277-294.

[52] Martin Godec, Peter Roth & Horst Bischof. (2012). Hough-based tracking of non-rigid objects. *Computer Vision and Image Understanding*, 117(10), 1245–1256.

[53] Shu Wang, Huchuan Lu, Fan Yang & Ming-Hsuan Yang. (2011). Superpixel tracking. *IEEE International Conference on Computer Vision (ICCV) 2011*, 1323-1330.

[54] Sam Hare, Amir Saffari & Philip Torr. (2015). Struck: Structured Output Tracking with Kernels. *IEEE International Conference on Computer Vision (ICCV) 2011*, 263-270.

[55] Michael Flynn. (1972). Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, C–21(9), 948–960.

[56] Zhipeng Zhang, Houwen Peng, Jianlong Fu, Bing Li & Weiming Hu. (2020). Ocean: Object-aware Anchor-free Tracking. *European Conference on Computer Vision 2020 (ECCV)*, 771-787.

[57] Lichao Zhang, Abel Gonzalez-Garcia, Joost van de Weijer, Martin Danelljan & Fahad Shahbaz Khan. (2019). Learning the Model Update for Siamese Trackers. *IEEE International Conference on Computer Vision (ICCV) 2019*, 4009-4018.

[58] Bin Yan, Houwen Peng, Jianlong Fu, Dong Wang & Huchuan Lu. (2021). Learning Spatio-Temporal Transformer for Visual Tracking. *IEEE International Conference on Computer Vision (ICCV) 2021*, 10448–10457.

[59] Goutam Bhat, Martin Danelljan, Luc Van Gool & Radu Timofte. (2020). Learning Discriminative Model Prediction for Tracking. *IEEE International Conference on Computer Vision (ICCV) 2019*, 6181-6190.

[60] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan & Michael Felsberg. (2019). ATOM: Accurate Tracking by Overlap Maximization. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2019*, 4655–4664.

[61] Chuan-Chi Wang, Chun-Yen Ho, Chia-Heng Tu & Shih-Hao Hung. (2022). cuPSO: GPU Parallelization for Particle Swarm Optimization Algorithms. *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 1183-1189.

[62] Dylan Janssen, Wayne Pullan & Alan Wee-Cung Liew. (2023). GPU Based Differential Evolution: New Insights and Comparative Study. *Concurrency Computation: Practice and Experience 2023,* 1-18.

[63] Jing Liu, Ruhul Sarker, Saber Elsayed, Daryl Essam & Nurhadi Siswanto. (2024). Large-scale evolutionary optimization: A review and comparative study. *Swarm Intelligence and Evolutionary Computation*, 85, 101466 – 101490.

[64] Ying Tan & Ke Ding. (2015). A survey on GPU-based implementation of swarm intelligence algorithms. *IEEE Transactions on Cybernetics*, 46(9), 2028–2041.

[65] Fevrier Valdez, Patricia Melin & Oscar Castillo. (2013). Bio-inspired Optimization Methods on Graphic Processing Unit for Minimization of Complex Mathematical Functions. *Recent Advances on Hybrid Intelligent Systems. Studies in Computational Intelligence*, 451, 313–322.

[66] Erick Cantú-Paz. (2001). *Efficient and Accurate Parallel Genetic Algorithms*. Springer.

[67] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan & Michael Felsberg. (2015). Convolutional Features for Correlation Filter Based Visual Tracking. *Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2015 Workshops*, 58–66.

[68] Clyde Kruskal & Carl Haerbert Smith. (1988). On the notion of granularity. *The Journal of Supercomputing*, 1, 395–408.

[69] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid & Silvio Savarese. (2019). Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2019*, 658-666.

[70] Yan Song, Zheng Hu, Tiancheng Li & Hongqi Fan. (2022). Performance Evaluation Metrics and Approaches for Target Tracking: A Survey. *Sensors*, 22(3), 793-813.

[71] Bolun Cai, Xiangmin Xu, Xiaofen Xing, Kui Jia, Jie Miao & Dacheng Tao. (2019). BIT: Biologically Inspired Tracker. *IEEE Transactions on Image Processing*, 25(3), 1327-1339.

[72] Yinzhe Xu, Huajian Huang, Yingshu Chen & Sai-Kit Yeung. (2024). 360VOTS: Visual Object Tracking and Segmentation in Omnidirectional Videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(11), 9785-9797.

[73] Shiyi Liang, Yifan Bai, Yihong Gong & Xing We. (2025). Autoregressive Sequential Pretraining for Visual Tracking. *Computer Vision and Pattern Recognition 2025*, 7254–7264.

[74] Lianghua Huang, Xin Zhao & Kaiqi Huang. (2019). GOT-10k: A Large High-Diversity Benchmark for Generic Object Tracking in the Wild. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43, 1562-1577.

[75] Jeonghun Lee & Kwang-il Hwang. (2021). YOLO with adaptive frame control for real-time object detection applications. *Multimedia Tool and Applications*, 81, 36375–36396.

[76] James Davis, Yi-Hsuan Hsieh & Hung-Chi Lee. (2015). Humans perceive flicker artifacts at 500 Hz. *Scientific Reports*, 5(1), 7861-7865.

[77] Michael Kalloniatis & Charles Luu. (2007). Temporal Resolution. *U.S. National Library of Medicine*.

[78] Peter White. (2018). Is conscious perception a series of discrete temporal frames?. *Consciousness and Cognition*, 60, 98–126.

[79] NVIDIA Corporation. (2025). *CUDA Toolkit*. NVIDIA Developer. https://developer.nvidia.com/cuda-toolkit

[80] The Kronos Group Inc. (2025). *OpenCL for Parallel Programming of Heterogeneous Systems*. Kronos Group. https://www.khronos.org/opencl/

[81] Kenneth Leroy Busbee. (2008). *Programming Fundamentals - A Modular Structured Approach using C++*. Connexions.

[82] Dorin Comaniciu, Visvanathan Ramesh & Peter Meer. (2003). Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5), 564-577.

[83] João Henriques, Rui Caseiro, Pedro Martins & Jorge Batista. (2015). High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3), 583–596.

[84] Luca Bertinetto, Jack Valmadre, João Henriques, Andrea Vedaldi & Philip Torr. (2021). Fully-Convolutional Siamese Networks for Object Tracking. *European Conference on Computer Vision (ECCV) 2016 Workshops*, 850-865.

[85] Matthias Müller, Adel Bibi, Silvio Giancola, Salman Al-Subaihi & Bernard Ghanem. (2018). *European Conference on Computer Vision (ECCV) 2018*, 300-317.

[86] Matthias Mueller, Neil Smith & Bernard Ghanem. (2016). A Benchmark and Simulator for UAV Tracking. *European Conference on Computer Vision (ECCV) 2016*, 445-461.

[87] Eun-bin An, Ayoung Kim, Soon-heung Jung, Sangwoon Kwak, Jin Young Lee, Won-Sik Cheong, Hyon-Gon Choo & Kwang-deok Seo. (2024). Adaptive spatial down-sampling method based on object occupancy distribution for video coding for machines. *EURASIP Journal on Image and Video Processing*, 2024.

[88] Joblib Developers. (2021). *Joblib: running Python functions as pipeline jobs*. Joblib. https://joblib.readthedocs.io/en/stable/

[89] Peter Christensen, David Hand, Nishadi Kirielle. (2023). A Review of the F-Measure: Its History, Properties, Criticism, and Alternatives. *ACM Computing Surveys*, 56(3), 1-24.

[90] Bolun Cai. (2015). A Perfect Tracker on CVPR TB-50 & ALOV300++. *School of Electronic and Information Engineering, South China University of Technology*.

[91] Víctor Alejandro Méndez-López, Carlos Soubervielle-Montalvo, Cesar Puente, Rafael Peña-Gallardo, Emilio Jorge González-Galván. (2025). Updating and Evaluating the Struck Tracker: A Comparative Study on the ALOV300++ Benchmark. *Research in Computing Science*, 153(10), 45–58.

Appendices

Appendix 1. Results of IoU thresholds tests for new exploration

Following table shows a comparison between the results with IoU < 0.6 and IoU < 0.5 as thresholds to execute simple exploration (as it is explained in section 4.4) in order to justify why 0.5 as that threshold:

Category	Video	IoU < 0.6		IoU < 0.5	
		F1-Score	FPS	F1-Score	FPS
01-Light	00001	40%	11.81005174	41%	12.1441841
	00002	97%	2.978625239	98%	3.11688288
	00003	82%	3.019495973	77%	3.0659568
	00004	59%	10.32053494	50%	9.85390823
	00005	55%	5.174841778	54%	5.6127362
	00006	40%	18.87766042	54%	18.9517621
	00007	70%	6.996679955	81%	7.41721572
	00008	22%	8.341193321	14%	7.56744956
	00009	78%	5.911881738	78%	5.8572198
	00010	33%	12.02924931	32%	12.0450515
	00011	48%	12.75458245	49%	12.9909198
	00012	8%	16.03927387	4%	16.3578412
	00013	42%	13.97598134	47%	14.330745
	00014	34%	11.07836015	42%	11.2676037
	00015	11%	19.47076009	1%	18.9213209
	00016	39%	15.14320192	60%	15.5304204
	00017	17%	15.58581184	11%	15.1163797
	00018	87%	14.99567984	87%	15.2491239
	00019	55%	15.21611732	73%	15.6615382
	00020	78%	15.97073849	83%	16.3345567
	00021	69%	13.28798967	70%	13.4672422
	00022	7%	10.57515508	6%	10.9507967
	00023	52%	11.69579214	59%	12.1825018
	00024	80%	9.515675564	73%	9.46178398

	00025	45%	19.47821391	55%	19.7810217
	00026	30%	17.71204885	38%	18.1609415
	00027	21%	9.56577466	21%	10.4179085
	00028	6%	11.26437384	3%	11.2891306
	00029	45%	17.95799835	57%	17.9961528
	00030	49%	12.19142136	48%	12.1792523
	00031	80%	12.24810346	81%	12.4184299
	00032	31%	6.522972036	42%	6.67776354
	00033	0%	9.592415276	0%	9.87950616
02-SurfaceCover	00001	98%	9.582170394	98%	9.64686556
	00002	82%	11.61162276	87%	11.6127355
	00003	0%	4.073999012	15%	4.32228086
	00004	85%	2.794212125	80%	2.80935265
	00005	17%	2.440727429	17%	2.48189313
	00006	67%	10.23795817	76%	10.3635898
	00007	97%	6.147622179	98%	6.13009392
	00008	81%	8.863980677	97%	8.98944343
	00009	88%	10.09167654	84%	10.2155909
	00010	45%	4.247970641	34%	4.36220177
	00011	3%	2.863494786	3%	2.90104075
	00012	15%	18.30076926	19%	18.9539865
	00013	100%	0.815225989	100%	0.83333864
	00014	84%	5.290653033	87%	5.53112667
	00015	30%	18.3185396	37%	18.9321603
03-Specularity	00001	69%	4.471065647	57%	4.44395413
	00002	71%	0.709046262	70%	0.72278806
	00003	37%	18.1535728	44%	19.0627397
	00004	33%	2.751708357	11%	2.59761803
	00005	7%	3.308263802	4%	3.38526018
	00006	64%	1.510830841	65%	1.58356296
	00007	57%	4.814410301	59%	4.93225102
	00008	89%	3.030651718	82%	3.15442685
	00009	100%	1.006507887	100%	0.958741
	00010	97%	0.836671296	96%	0.82177083
	00011	66%	10.73598251	57%	10.3793822
	00012	24%	8.552201305	24%	9.13747701
	00013	22%	0.512257563	22%	0.51754534
	00014	26%	17.49459978	25%	18.0650821
	00015	51%	3.607320161	45%	3.88248063
	00016	78%	3.80731218	88%	4.18614831

	00017	65%	5.103122612	73%	5.49428288
	00018	87%	2.922221892	98%	3.0313049
04-Transparency	00001	47%	2.758661497	34%	2.89114847
	00002	89%	3.344137473	82%	3.45976294
	00003	60%	12.35034419	65%	12.3028726
	00004	31%	12.97040181	46%	13.867257
	00005	8%	19.02207788	3%	20.1456374
	00006	75%	11.39424187	68%	11.5928536
	00007	87%	3.153191023	97%	3.31577014
	00008	100%	6.562092009	100%	7.00245771
	00009	34%	11.22030208	34%	11.5851056
	00010	55%	10.07681995	62%	10.6374878
	00011	16%	10.21349295	17%	10.6621963
	00012	64%	9.15375322	70%	10.0358186
	00013	79%	6.872880527	86%	6.76709929
	00014	88%	14.31767008	79%	14.8715581
	00015	35%	5.327028971	33%	5.32270759
	00016	100%	8.297235648	100%	8.52672949
	00017	100%	3.444530673	100%	3.50962962
	00018	94%	3.262053273	96%	3.33374597
	00019	75%	12.94577479	65%	12.5753405
	00020	65%	17.59775151	47%	16.8416967
05-Shape	00001	0%	6.364901409	0%	6.2827397
	00002	24%	3.313451464	24%	3.24494521
	00003	24%	12.45537778	22%	12.5162107
	00004	39%	5.431217291	65%	5.65579857
	00005	59%	5.314570416	57%	5.36367948
	00006	14%	15.22248526	12%	15.5562188
	00007	59%	5.675576377	54%	5.8208434
	00008	79%	0.527094215	79%	0.52159284
	00009	42%	0.602709404	43%	0.61644764
	00010	33%	4.879390356	34%	5.11538601
	00011	8%	2.527720056	13%	2.73835264
	00012	28%	12.87273712	25%	12.4358818
	00013	23%	9.664958967	36%	9.64756027
	00014	75%	4.094219767	62%	3.94861804
	00015	87%	4.226065805	91%	4.10700937
	00016	82%	3.513828033	60%	3.34256213
	00017	79%	3.12145989	82%	3.03888884
	00018	73%	2.842630859	53%	2.67895695

	00019	62%	6.302670632	58%	6.25629985
	00020	29%	5.270873869	27%	5.20359336
	00021	0%	5.011325259	0%	4.99430649
	00022	35%	12.43024637	51%	12.522354
	00023	58%	3.506959525	62%	3.89453241
	00024	30%	0.475398651	31%	0.47732429
06-MotionSmoothness	00001	35%	11.57736107	27%	11.5642685
	00002	57%	21.00962495	66%	21.5250705
	00003	100%	0.629757816	100%	0.64742569
	00004	29%	7.876216484	29%	7.85883251
	00005	22%	14.96986494	18%	15.1486982
	00006	2%	5.251608705	0%	5.29764794
	00007	0%	14.51057626	0%	14.4650919
	00008	37%	9.90527945	24%	9.81778044
	00009	98%	3.404504023	97%	3.3913809
	00010	24%	6.940669681	24%	6.92609417
	00011	0%	2.848060002	0%	2.8478043
	00012	91%	8.836171069	100%	8.99249131
	00013	72%	1.418342655	65%	1.37646707
	00014	0%	11.52194584	0%	11.4879768
	00015	38%	10.70107654	47%	11.4244905
	00016	54%	13.78892613	76%	14.0823861
	00017	60%	13.91766334	81%	13.6520144
	00018	34%	3.941328567	33%	4.03456043
	00019	48%	17.68898379	39%	17.6174478
	00020	97%	7.781342941	95%	7.65671608
	00021	31%	20.42579292	48%	20.5155522
	00022	30%	7.587142065	30%	7.64079055
07-MotionCoherence	00001	65%	5.771104398	79%	6.09756796
	00002	15%	20.27016724	12%	20.3191287
	00003	35%	20.74015613	53%	21.1090598
	00004	1%	8.056889943	3%	8.79279936
	00005	0%	18.43461206	0%	18.6381575
	00006	43%	9.942272873	42%	10.0429359
	00007	63%	2.427824568	68%	2.43286102
	00008	47%	11.09570014	66%	11.3896599
	00009	29%	2.995632293	45%	3.10896802
	00010	53%	17.84351111	47%	18.0852359
	00011	33%	18.62785218	39%	18.8096816
	00012	27%	13.85233112	34%	14.0188036

	00001	46%	16.04350889	49%	16.2200364
	00002	73%	15.16682458	79%	15.4477894
	00003	75%	11.38735888	83%	11.5610315
	00004	56%	9.824858846	60%	9.99442128
	00005	78%	10.39237282	87%	10.6825681
	00006	32%	12.46984594	31%	12.4422862
	00007	100%	1.567369046	100%	1.57815941
08-Clutter	00008	32%	17.28487164	32%	17.31279
	00009	57%	14.91675192	90%	15.535579
	00010	40%	9.42102358	33%	9.80268457
	00011	66%	11.86224247	69%	11.9644887
	00012	86%	10.95056049	82%	11.0502011
	00013	64%	12.33928458	68%	12.3337333
	00014	17%	11.76404508	21%	11.9644341
	00015	20%	9.9647233	33%	10.2890847
	00001	0%	1.137092321	0%	1.1381618
	00002	61%	11.06263807	56%	11.8720638
	00003	47%	14.00131232	57%	14.3471253
	00004	39%	18.29682976	34%	18.738087
	00005	39%	9.778015845	32%	9.93306463
	00006	76%	1.029139662	76%	1.04520827
	00007	40%	7.155397154	14%	6.54436156
	00008	90%	14.58114819	71%	14.1782652
	00009	39%	7.774811211	68%	8.81395827
	00010	88%	6.057485497	87%	6.10384521
	00011	100%	1.69888675	89%	1.70783547
	00012	56%	4.854191398	47%	4.81285415
09-Confusion	00013	0%	18.90227427	0%	18.8548386
	00014	2%	12.30510012	5%	12.3351631
	00015	24%	1.236449856	25%	1.21311951
	00016	38%	10.62688024	57%	11.4016235
	00017	70%	10.32202151	64%	10.2183211
	00018	16%	17.96462325	7%	17.8260259
	00019	12%	18.82759124	13%	18.8926502
	00020	30%	12.37462421	77%	13.963265
	00021	66%	14.23561505	63%	14.4515089
	00022	25%	11.64031875	40%	12.5530623
	00023	80%	8.591226107	84%	8.91721197
	00024	2%	19.7266438	4%	19.6959481
	00025	32%	18.03338741	39%	18.0922146

	00026	39%	15.32268939	55%	15.3489727
	00027	49%	8.414700812	75%	8.70624157
	00028	33%	7.368533761	52%	7.76605715
	00029	59%	8.273761262	72%	8.37866657
	00030	69%	7.471646466	72%	7.59823219
	00031	95%	3.768222578	90%	3.79774858
	00032	25%	13.678879	29%	13.5467874
	00033	40%	15.32490615	61%	15.5867455
	00034	80%	6.689763681	81%	6.64463019
	00035	100%	5.915637402	100%	5.96957085
	00036	4%	10.72509766	7%	10.5340698
	00037	43%	6.780593791	35%	6.87156851
10-LowContrast	00001	22%	13.11622246	28%	12.9880235
	00002	80%	6.265271426	78%	6.5128585
	00003	91%	6.2877149	95%	6.36562703
	00004	29%	9.710258955	32%	9.83062705
	00005	81%	3.012926704	81%	3.02185824
	00006	68%	15.05995851	49%	14.9072736
	00007	57%	15.45637495	68%	15.4660662
	00008	61%	11.0889989	57%	11.2157093
	00009	59%	13.77313535	53%	14.4398587
	00010	92%	5.157732771	84%	5.19334464
	00011	35%	12.75344211	42%	13.2676583
	00012	83%	0.725319271	92%	0.72128333
	00013	67%	2.581001385	66%	2.58902112
	00014	26%	4.336863335	28%	4.35782427
	00015	73%	6.786954038	75%	6.99322312
	00016	69%	10.6028521	65%	10.7813629
	00017	34%	11.04331233	40%	10.7415416
	00018	48%	9.67562576	22%	9.24400329
	00019	22%	12.17943619	28%	12.3731609
	00020	12%	8.535296712	18%	8.88687268
	00021	72%	8.937298137	66%	8.78552079
	00022	42%	12.78498903	43%	13.0869118
	00023	92%	0.671497976	97%	0.67200657
11-Occlusion	00001	14%	4.551794784	6%	4.49983544
	00002	6%	12.22895387	6%	12.2514746
	00003	38%	7.044340975	24%	6.94823549
	00004	35%	8.113311915	35%	8.18497662
	00005	12%	3.323424636	11%	3.30400852

	00006	49%	6.873568918	56%	7.13892723
	00007	5%	6.4116874	31%	7.17782784
	00008	32%	12.39563265	37%	12.2940688
	00009	100%	14.79312359	97%	14.8369969
	00010	29%	14.05838138	42%	14.5923805
	00011	54%	1.657187641	56%	1.68875768
	00012	0%	13.41601876	0%	13.338474
	00013	77%	5.52229275	63%	6.19170152
	00014	63%	4.951951927	63%	5.27200481
	00015	96%	4.369042954	99%	4.42559557
	00016	41%	0.595523383	77%	0.67556479
	00017	45%	5.208206587	62%	5.30950187
	00018	37%	0.580202771	37%	0.55676704
	00019	40%	7.803934137	46%	7.83382084
	00020	78%	18.44464152	71%	17.871467
	00021	87%	0.660786832	85%	0.67709174
	00022	19%	2.840543505	13%	2.78529083
	00023	79%	8.749105955	69%	8.61630135
	00024	44%	11.58814724	47%	12.3481693
	00025	74%	11.88078171	80%	12.0653261
	00026	51%	13.15501928	36%	12.6718119
	00027	56%	13.37252665	66%	13.4727694
	00028	18%	11.58732678	19%	11.5469065
	00029	23%	8.711860778	23%	8.96980769
	00030	69%	10.42834706	47%	10.1773268
	00031	30%	6.817732966	23%	7.11754532
	00032	75%	10.58599679	58%	9.35787664
	00033	20%	12.50248556	16%	12.4853764
	00034	0%	5.872378971	0%	6.07786462
12-MovingCamera	00001	21%	3.312189902	37%	3.55569711
	00002	93%	1.537859196	93%	1.55135451
	00003	13%	9.127771467	10%	8.970363
	00004	39%	9.692584147	25%	9.16343087
	00005	43%	0.552134111	42%	0.54755873
	00006	58%	8.281149054	25%	7.69566808
	00007	51%	14.5179586	64%	14.4535527
	00008	45%	8.089128517	46%	7.88386942
	00009	62%	5.236533024	75%	5.40773798
	00010	0%	4.978797236	0%	4.98029591
	00011	56%	10.00874508	73%	10.1759272

	00012	49%	10.84742906	77%	11.2908329
	00013	49%	5.405755807	48%	5.46281954
	00014	85%	6.91387652	86%	6.84636042
	00015	76%	8.279870131	80%	9.10558744
	00016	50%	9.071795521	42%	9.04308084
	00017	71%	18.68181085	63%	18.1655554
	00018	79%	14.96294186	76%	14.6189608
	00019	76%	19.9553403	68%	19.4131926
	00020	73%	12.57039598	79%	12.2113011
	00021	71%	10.68205345	68%	10.3233734
	00022	14%	12.41643622	28%	12.7649961
	00001	92%	0.887538662	92%	0.89187568
	00002	5%	16.99595313	7%	16.4314818
	00003	6%	16.46046252	6%	16.0405225
	00004	5%	17.14950866	7%	16.6001128
	00005	13%	16.1058798	12%	15.5040483
	00006	26%	9.425483823	30%	8.71153021
	00007	55%	10.00941295	64%	10.0741117
	00008	62%	4.776503396	63%	4.71992603
	00009	14%	16.48472648	15%	16.1286245
	00010	36%	10.78691999	56%	10.9124639
	00011	32%	5.447455353	41%	5.76380151
	00012	20%	17.10773829	14%	16.4826381
	00013	53%	3.117210328	59%	3.21598061
	00014	14%	17.16929192	22%	16.7793815
13-ZoomingCamera	00015	27%	9.368583869	46%	9.28381565
	00016	35%	6.694545793	37%	6.65701795
	00017	25%	1.280532983	25%	1.28127635
	00018	6%	7.860260054	12%	8.04977527
	00019	45%	0.55789746	42%	0.52518304
	00020	76%	3.887072426	72%	3.96892525
	00021	27%	14.03013238	53%	14.5076579
	00022	15%	17.70536832	26%	17.1544732
	00023	70%	11.84357539	76%	11.4522808
	00024	25%	8.49883825	17%	8.22740284
	00025	71%	5.179117613	69%	5.1118745
	00026	66%	5.547850258	67%	5.34733243
	00027	31%	15.34147432	32%	14.65948
	00028	63%	11.55623802	51%	11.1032512
	00029	38%	10.55965154	40%	10.6544579

14-LongDuration	00001	2%	19.18514755	2%	18.4337448
	00002	13%	18.87927436	10%	18.1468879
	00003	11%	18.16822035	18%	17.3787084
	00004	32%	3.615957014	33%	3.44067336
	00005	5%	10.4601618	10%	10.0388727
	00006	61%	1.154405733	82%	1.18721959
	00007	18%	0.549440328	30%	0.51231992
	00008	13%	6.385353678	15%	6.04606441
	00009	14%	6.404058698	25%	6.29246101
	00010	2%	8.842368624	19%	9.17680298
Average		48%	9.306924589	49%	9.37913508

Appendix 2. Tests to adjust the automatic population

Following table shows the results from tests made in order to adjust the population sizes according to the search radius ranges. First section of the table shows the first results and sizes according to the first hypothetical configuration that was formulated, second section shows the results by increasing the population. First, it shows the population increase that helped improve the F1-Score, but by a negligible percentage. Then, it shows the case where there was a significant increase. Boxes marked NA indicate two possible scenarios: the video with the standard settings already achieved a 100% F1-Score, or there was no increase at all even with the maximum population size. The third section similarly shows the tests where the population was reduced to see how much F1-Score decreased. In this case, the boxes marked NA indicate that the video already obtained an F1-Score of 0% even with the normal settings.

Category	**Video**	**Search radius**	**Usual case**		
			Population	**F1-Score**	**FPS**
01-Light	00001	249	1024	43%	4.75
	00002	359	4096	100%	1.56
	00004	276	1024	50%	4.34
	00007	240	2048	62%	4.48
02-SurfaceCover	00001	180	1024	92%	5.67
	00002	180	1024	88%	5.2
	00012	120	512	7%	13.5
03-Specularity	00001	359	4096	63%	1.82

	00003	120	512	65%	12.9
	00004	540	4096	20%	0.48
04-Transparency	00002	360	4096	96%	1.84
	00003	180	1024	72%	4.64
05-Shape	00001	360	2048	0%	1.98
	00002	360	4096	23%	1.11
06-MotionSmoothness	00001	199	512	55%	6.72
	00002	80	512	0%	7.56
07-MotionCoherence	00001	240	2048	68%	2.64
	00006	228	512	25%	3.38
08-Clutter	00001	139	512	72%	9.36
	00002	184	512	83%	9.42
09-Confusion	00001	540	8192	0%	0.28
	00002	244	1024	73%	5.01
10-LowContrast	00002	240	2048	77%	3.26
	00004	251	1024	45%	3.67
11-Occlusion	00001	401	2048	23%	1.2
	00004	360	1024	30%	2.76
12-MovingCamera	00002	540	8192	93%	0.51
	00004	341	1024	39%	4.07
	00007	247	1024	55%	5.73
13-ZoomingCamera	00008	406	2048	47%	1.73
	00009	144	512	9%	9.93
	00010	234	1024	36%	5.15
14-LongDuration	00001	59	512	3%	10.08

Population increase tests

Category	Video	Point of F1-Score increase (population)	Point of drastic F1-Score increase		
			Population	F1-Score	FPS
01-Light	00001	2048	8192	50%	1.1
	00002	NA	NA	NA	NA
	00004	2048	8192	76%	0.83
	00007	4096	8192	100%	1.7
02-SurfaceCover	00001	2048	2048	100%	3.47
	00002	2048	2048	93%	3.39
	00012	1024	2048	34%	6.36
03-Specularity	00001	16384	16384	70%	0.4
	00003	NA	NA	NA	NA
	00004	16384	16384	54%	0.31

04-Transparency	00002	8192	8192	100%	0.91
	00003	2048	2048	92%	4.21
05-Shape	00001	NA	NA	NA	NA
	00002	NA	NA	NA	NA
06-MotionSmoothness	00001	8192	8192	81%	0.81
	00002	1024	1024	88%	8.83
07-MotionCoherence	00001	4096	4096	97%	1.97
	00006	1024	2048	52%	3.67
08-Clutter	00001	2048	2048	91%	10.2
	00002	1024	1024	92%	4.44
09-Confusion	00001	NA	NA	NA	NA
	00002	4096	4096	97%	3.63
10-LowContrast	00002	4096	4096	90%	5.54
	00004	4096	8193	NA	0.65
11-Occlusion	00001	4096	4096	41%	0.6
	00004	2048	8193	40%	0.68
12-MovingCamera	00002	NA	NA	NA	NA
	00004	NA	NA	NA	NA
	00007	2048	2048	98%	4.75
13-ZoomingCamera	00008	4096	4096	66%	1.54
	00009	1024	4096	41%	3.65
	00010	2048	2048	53%	3.85
14-LongDuration	00001	NA	NA	NA	NA

Population decrease tests

Category	Video	Point of decrease in F1-Score (population)	Point of drastic F1-Score decrease		
			Population	F1-Score	FPS
01-Light	00001	512	128	0%	5.72
	00002	2048	256	53%	2.42
	00004	256	128	15%	4.14
	00007	1024	1024	46%	5.72
02-SurfaceCover	00001	512	256	71%	8.88
	00002	512	512	52%	2.23
	00012	256	128	0%	21
03-Specularity	00001	1024	512	0%	3.48
	00003	256	256	13%	16.89
	00004	512	512	0%	0.58
04-Transparency	00002	256	128	44%	4.59
	00003	256	256	47%	11.27
05-Shape	00001	NA	NA	NA	NA

	00002	512	64	0%	2.19
06-MotionSmoothness	00001	256	256	33%	8.48
	00002	NA	NA	NA	NA
07-MotionCoherence	00001	1024	512	19%	5.74
	00006	32	32	6%	2.78
08-Clutter	00001	128	128	35%	12.16
	00002	256	256	25%	8.15
09-Confusion	00001	NA	NA	NA	NA
	00002	512	512	44%	5.95
10-LowContrast	00002	1024	1024	57%	4.54
	00004	512	512	28%	5.45
11-Occlusion	00001	1024	1024	0%	1.36
	00004	512	512	0%	2.65
12-MovingCamera	00002	512	256	56%	0.41
	00004	512	128	0%	6.86
	00007	512	128	26%	9.38
13-ZoomingCamera	00008	1028	256	7%	0.07
	00009	128	64	0%	13.4
	00010	512	512	11%	8.06
14-LongDuration	00001	128	64	0%	14.09

Appendix 3. Results of evolutionary operator tests

Following table shows the results of gotten by combining different evolutionary operators and demonstrate why the operators described in section 4.4.1 were chosen:

F1-Score

Operators	**Selection**	**Roulette**								
	Crossover	**Uniform**			**Blend**			**SBX**		
	Mutation	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**
Category	**Video**									
01-Light	00001	36%	10%	28%	10%	61%	36%	43%	19%	19%
	00002	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00004	0%	15%	0%	22%	0%	22%	7%	7%	15%
02-SurfaceCover	00001	77%	77%	72%	74%	53%	15%	55%	15%	36%
	00002	38%	48%	95%	27%	33%	48%	52%	61%	52%
	00008	72%	33%	60%	80%	66%	46%	33%	12%	37%
03-Specularity	00001	0%	14%	0%	0%	0%	14%	0%	0%	0%
	00003	0%	3%	6%	0%	0%	10%	0%	10%	6%

04-Transparency	00002	10%	17%	32%	26%	17%	49%	57%	29%	17%
	00003	0%	0%	7%	7%	0%	7%	10%	3%	28%
05-Shape	00001	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00013	0%	0%	9%	0%	9%	5%	9%	0%	5%
06-MotionSmoothness	00002	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00003	26%	18%	46%	74%	29%	41%	33%	70%	90%
	00004	28%	40%	28%	28%	0%	28%	15%	0%	28%
07-MotionCoherence	00001	31%	23%	31%	19%	31%	14%	31%	27%	34%
	00006	19%	19%	19%	19%	25%	44%	25%	25%	19%
	00008	0%	0%	0%	0%	0%	0%	0%	0%	0%
08-Clutter	00002	0%	35%	13%	0%	0%	0%	0%	0%	0%
	00004	3%	2%	9%	3%	0%	7%	3%	3%	2%
09-Confusion	00001	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00002	0%	0%	0%	0%	0%	5%	10%	0%	5%
10-LowContrast	00002	16%	8%	13%	16%	38%	5%	13%	18%	49%
11-Occlusion	00004	0%	0%	0%	0%	0%	0%	0%	0%	6%
	00006	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00016	25%	25%	32%	50%	32%	17%	32%	32%	32%
12-MovingCamera	00002	8%	0%	8%	0%	0%	0%	0%	0%	0%
	00004	0%	5%	0%	0%	0%	0%	0%	0%	0%
	00007	14%	55%	79%	20%	20%	32%	66%	14%	32%
13-ZoomingCamera	00008	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00009	0%	0%	0%	0%	0%	0%	0%	0%	0%
14-LongDuration	00001	0%	0%	1%	0%	0%	0%	0%	0%	0%
	00006	1%	0%	0%	1%	0%	0%	0%	1%	0%
Average		12%	14%	18%	14%	13%	13%	15%	10%	16%

Operators	**Selection**	**Tournament**								
	Crossover	**Uniform**			**Blend**			**SBX**		
	Mutation	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**
Category	**Video**									
01-Light	00001	50%	0%	43%	0%	61%	61%	38%	50%	80%
	00002	100%	100%	100%	100%	100%	84%	100%	100%	100%
	00004	70%	54%	70%	70%	70%	70%	66%	70%	70%
02-SurfaceCover	00001	100%	100%	100%	100%	100%	100%	100%	100%	100%
	00002	86%	86%	86%	86%	86%	86%	86%	86%	86%
	00008	98%	98%	98%	98%	98%	98%	98%	98%	98%
03-Specularity	00001	70%	70%	70%	70%	70%	47%	37%	70%	70%
	00003	50%	51%	60%	96%	51%	51%	50%	51%	51%

04-Transparency	00002	100%	100%	100%	100%	100%	100%	100%	100%	100%
	00003	45%	75%	75%	75%	96%	97%	75%	74%	71%
05-Shape	00001	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00013	30%	30%	30%	30%	30%	30%	30%	30%	30%
06-MotionSmoothness	00002	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00003	100%	100%	100%	100%	100%	100%	100%	100%	100%
	00004	28%	28%	28%	28%	28%	28%	28%	28%	28%
07-MotionCoherence	00001	38%	38%	38%	38%	38%	38%	38%	38%	38%
	00006	33%	35%	4%	52%	48%	52%	40%	50%	52%
	00008	40%	82%	46%	100%	100%	100%	100%	5%	30%
08-Clutter	00002	88%	88%	83%	88%	88%	88%	83%	88%	78%
	00004	80%	95%	90%	88%	98%	100%	77%	81%	86%
09-Confusion	00001	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00002	73%	83%	100%	100%	100%	100%	66%	64%	71%
10-LowContrast	00002	84%	86%	86%	85%	86%	86%	86%	86%	86%
11-Occlusion	00004	40%	40%	40%	40%	40%	40%	40%	40%	40%
	00006	5%	5%	5%	5%	5%	5%	5%	5%	5%
	00016	100%	32%	32%	36%	32%	32%	32%	32%	100%
12-MovingCamera	00002	8%	28%	8%	36%	8%	8%	8%	8%	8%
	00004	26%	19%	24%	10%	28%	28%	20%	30%	20%
	00007	86%	5%	100%	90%	100%	100%	81%	100%	100%
13-ZoomingCamera	00008	63%	63%	46%	66%	63%	97%	66%	66%	66%
	00009	68%	45%	56%	31%	24%	38%	3%	34%	24%
14-LongDuration	00001	0%	0%	0%	0%	0%	0%	0%	0%	0%
	00006	90%	28%	49%	86%	98%	100%	77%	72%	76%
Average		56%	50%	54%	58%	59%	60%	52%	53%	56%

FPS

Operators	**Selection**	**Roulette**								
	Crossover	**Uniform**			**Blend**			**SBX**		
	Mutation	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**
Category	**Video**									
01-Light	00001	3.29	3.15	3.37	3.48	3.5	2.99	3.14	3.02	3.05
	00002	1.23	1.3	1.29	1.3	1.69	1.32	1.37	1.41	1.34
	00004	2.08	3.59	3.65	3.2	2.15	2.64	2.69	3.77	3.03
02-SurfaceCover	00001	2.68	2.77	2.81	2.76	2.44	3.25	2.45	3.17	3.26
	00002	2.84	2.7	2.74	1.83	2.69	2.82	3.15	1.47	2.52
	00008	1.95	2.73	3.05	1.67	3.3	2.75	3.37	3.06	2.33
03-Specularity	00001	1.5	1.47	2.42	2.68	2.91	2.33	2.68	1.17	2.05

	00003	3.77	2.86	5.15	4.45	4.61	2.96	2.25	4.44	3.42
04-Transparency	00002	1.56	1.64	1.67	1.75	2.05	1.38	1.4	2	1.6
	00003	2.34	1.88	2.8	3.83	3.05	2.49	4.13	4.44	2.86
05-Shape	00001	1.41	1.07	2.43	1.99	2.55	2.06	2.79	3.06	2.07
	00013	2	2.88	2.01	2.3	3.23	2.03	2.99	2.96	2.05
06-MotionSmoothness	00002	3.2	3	1.3	1.87	2.2	1.99	2.4	1.8	2.36
	00003	0.51	0.52	0.46	0.38	0.53	0.4	0.47	0.48	0.41
	00004	2.34	1.83	2.16	1.31	2.81	2.31	2.33	2.64	1.9
07-MotionCoherence	00001	2.93	1.57	1.79	1.87	2.29	2.79	2.49	2.41	1.92
	00006	2.27	1.68	1.42	1.9	2.15	2.17	2.24	2.27	2.05
	00008	3.16	2.2	1.43	3.04	2.58	3.04	3.28	3.18	2.77
08-Clutter	00002	3.65	2.18	3.41	2.21	3.42	3.41	3.76	3.91	2.56
	00004	2.88	2.67	2.34	1.59	3.11	2.71	2.22	1.71	2.34
09-Confusion	00001	1.11	1.14	1.17	0.85	1.16	1.08	1.13	1.09	1.17
	00002	3.16	3.01	2.15	1.86	3.1	2.8	3.09	2.41	2.57
10-LowContrast	00002	1.92	1.99	1.84	1.6	2.92	2.74	1.66	1.61	1.74
11-Occlusion	00004	2.53	1.82	1.94	2.23	2.31	2.2	1.82	2.28	2.04
	00006	1.81	1.58	1.59	1.49	1.56	2.7	2.27	2.26	1.56
	00016	0.71	0.61	0.62	0.59	0.55	0.69	0.7	0.74	0.56
12-MovingCamera	00002	1.06	0.95	0.87	0.9	0.98	1.07	0.7	1.02	1.1
	00004	3	2.37	2.26	2.1	3.08	3.06	2.46	3.31	3.3
	00007	3.42	2.32	2.16	1.92	2.22	3.49	2.37	3.71	3.5
13-ZoomingCamera	00008	1.33	1.1	1.57	0.95	0.96	1.4	1.48	1.78	1.41
	00009	4.38	2.93	4.41	2.74	4.46	4.4	3.36	4.48	4.46
14-LongDuration	00001	2.93	4.47	3.16	3	4.63	4.49	4.77	4.51	4.65
	00006	1.14	1.02	1.14	1.07	1.1	1.11	1.16	0.94	1.19
Average		2.31	2.09	2.20	2.02	2.49	2.40	2.38	2.50	2.28

Operators	**Selection**	**Tournament**								
	Crossover	**Uniform**			**Blend**			**SBX**		
	Mutation	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**	**Unif.**	**Poly.**	**Gauss.**
Category	**Video**									
01-Light	00001	2.35	2.86	2.85	1.75	1.96	1.98	2.01	2.05	2.04
	00002	1.27	1.3	1.35	1.01	1.02	1.03	1.06	1.12	1.12
	00004	2.7	2.7	2.75	1.86	1.86	1.88	2.03	2.02	2.04
02-SurfaceCover	00001	2.16	2.82	2.86	1.5	1.85	1.87	1.95	1.72	1.75
	00002	2.72	2.83	2.33	1.49	1.61	1.58	1.66	1.86	1.68
	00008	2.05	2.38	2.49	1.68	1.71	1.42	1.63	1.89	1.94
03-Specularity	00001	1.85	2.36	2.39	1.31	1.74	1.3	1.83	1.38	1.23

	00003	3.72	3.69	4.14	1.03	2.31	2.07	2.37	2.65	2.72
04-Transparency	00002	1.82	1.83	2.05	1.09	1.55	1.26	1.47	1.7	1.68
	00003	2.24	3.51	3.61	1.77	2.23	1.77	2.41	2.11	1.74
05-Shape	00001	1.84	2.73	2.01	1.34	1.86	1.42	1.97	1.61	1.92
	00013	2.14	2.37	2.64	1.58	1.78	1.41	1.67	1.98	1.98
06-MotionSmoothness	00002	2.33	2.5	1.68	1.2	1.6	1.25	1.71	1.22	1.73
	00003	0.36	0.5	0.51	0.47	0.48	0.49	0.44	0.5	0.5
	00004	2.47	2.46	2.09	1.6	1.33	1.84	1.38	1.87	1.78
07-MotionCoherence	00001	2.19	2.23	1.83	1.56	1.58	1.32	1.22	1.37	1.68
	00006	0.33	2.39	2.19	1.74	1.53	1.77	1.58	1.83	1.87
	00008	2.75	2.96	2.74	2.14	1.91	2.09	1.94	2.17	2.18
08-Clutter	00002	3.23	3.39	2.62	2.23	2.3	1.86	2.05	1.98	2.39
	00004	2.7	2.57	2.49	1.9	0.98	1.83	1.67	1.46	1.82
09-Confusion	00001	1.62	1.58	1.4	1.24	1.05	1.24	1.01	1.04	1.03
	00002	3.06	3.12	2.44	2.17	2.16	1.84	1.54	1.86	2.25
10-LowContrast	00002	2.26	2.19	1.35	1.22	1.38	1.25	1.43	1.42	1.4
11-Occlusion	00004	2.58	2.3	2.32	1.22	1.49	1.51	1.43	1.48	1.5
	00006	2.17	2.09	2.06	1.29	1.46	1.36	1.16	1.41	1.39
	00016	0.59	0.67	0.64	0.54	0.55	0.56	0.61	0.58	0.55
12-MovingCamera	00002	0.59	3.64	1.11	2.17	0.61	0.8	0.94	0.98	0.93
	00004	2.94	3.8	3.02	1.96	1.25	1.78	2.02	2.12	2.06
	00007	2.93	2.45	3.06	1.7	1.93	1.79	1.97	2.13	2.09
13-ZoomingCamera	00008	1.78	1.81	2.38	1.22	1.31	1.04	1.31	0.83	1.37
	00009	3.69	3.72	1.89	2.2	2.35	2.08	2.43	2.48	2.4
14-LongDuration	00001	4.16	4.3	4.58	2.32	1.96	2.15	2.6	2.69	2.64
	00006	1.11	1.1	0.97	0.84	0.96	0.81	0.92	0.88	0.98
Average		2.20	2.52	2.27	1.53	1.57	1.50	1.62	1.65	1.71