Mohamed Khider University of Biskra
Faculty of Science and Technology
Department of Electrical Engineering

MASTER THESIS

Electrical Engineering
Telecommunication
Networks and Telecommunication

Réf. : ……………………………………………

Submitted and Defended by:
AMMARI Abdessalem

On: Monday, June 27, 2022

OpenCV Object Tracking

Board of Examiners:

University year: 2021 - 2022

Mohamed Khider University of Biskra
Faculty of Science and Technology
Department of Electrical Engineering

MASTER THESIS

Electrical Engineering
Telecommunication
Networks and Telecommunication

Réf. : ……………………………………………

Submitted and Defended by:
AMMARI Abdessalem

On: Monday, June 27, 2022

OpenCV Object Tracking

In:…………………………………

Presented by: **Favorable opinion of the supervisor:**

AMMARI ABD ESSALEM **Dr. MEDOUAKH Saadia**

Favorable opinion of the jury president

TOUMI ABIDA

Stamp and signature

SUMMARY (English and Arabic)

Abstract:

Object tracking is one of the most important and fundamental disciplines of Computer Vision. Many Computer Vision applications require specific object tracking capabilities, including autonomous and smart vehicles, video surveillance, medical treatments, and many others. The OpenCV as one of the most popular libraries for Computer Vision includes several hundred Computer Vision algorithms. Object tracking tasks in the library can be roughly clustered in single and multiple object trackers. The library is widely used for real-time applications, but there are a lot of unanswered questions such as when to use a specific tracker, how to evaluate its performance, and for what kind of objects will the tracker yield the best results? In this thesis, we experiment with single object tracking with 3 trackers implemented in OpenCV against the OTB dataset. The results are shown based on quantitative (Center location error and Overlap rate metrics) and qualitative (visually).

Key words: Computer Vision, Object Tracking, single Object Tracking, OpenCV

ملخص :

يعد تتبع الكائنات أحد أهم التخصصات الأساسية لرؤية الكمبيوتر. تتطلب العديد من تطبيقات رؤية الكمبيوتر قدرات محددة لتتبع الكائنات، بما في ذلك المركبات الذكية وذاتية القيادة، والمراقبة بالفيديو، والعلاجات الطبية، وغيرها الكثير. تتضمن OpenCV كواحدة من أكثر المكتبات شعبية لرؤية الكمبيوتر عدة مئات من خوارزميات رؤية الكمبيوتر. يمكن تجميع مهام تتبع الكائنات في المكتبة تقريبًا في أجهزة تعقب كائنات واحدة ومتعددة. تُستخدم المكتبة على نطاق واسع للتطبيقات في الوقت الفعلي، ولكن هناك الكثير من الأسئلة التي لم تتم الإجابة عليها مثل متى يجب استخدام متتبع معين ، وكيفية تقييم أدائه ، ولأي نوع من الكائنات سيحقق المتعقب أفضل النتائج؟ في هذه الأطروحة ، نجرب تتبع كائن واحد باستخدام 3 أدوات تعقب تم تنفيذها في OpenCV مقابل مجموعة بيانات OTB. يتم عرض النتائج بناءً على الكمية (خطأ موقع المركز ومقاييس معدل التداخل) والنوعية (بصريًا).

الكلمات المفتاحية : رؤية الكمبيوتر ، تتبع الكائن ، تتبع كائن واحد ، OpenCV

Acknowledgment

Alhamdulillah, I praise and thank Allah Almighty, for giving me the strength and courage to complete this work.

I would like to express my gratitude and sincere thanks to my supervisor MEDOUAKH SAADIA for her guidance, encouragement, and valuable remarks throughout the preparation of this master thesis.

I would like to thank the members of the jury:

Dr. TOUMI Abida, for accepting to be president of the jury.

Dr. FEDIAS Meriem, for accepting to examine our work.

Sincere thanks to all the teachers who taught me during all my studies.

I would also like to thank all the people who have encouraged me during the realization of this work, family, colleagues, and friends.

Dedication

First and foremost, I thank Allah Almighty for the strength and patience he has given me to accomplish this work.

I dedicate this work to my father "Allah Yarhmo", and to my mother who helped me in every step of my life and gave me the willingness to overcome everything in life.

To all of my brothers and sisters who always encourage me.

To all my friends, and especially my best friend Yacine.

Table of contents

List of figures

List of tables

General Intro-duction

General introduction

Computer vision is a rapidly growing interdisciplinary scientific field devoted to analyzing, modifying, and high-level understanding of images. It has been a subject of increasing interest for the last two decades. Its popularity stems from the fact that it provides the means to see as humans do [1], and in some applications it can even outperform humans [2] [3]. Object tracking is one of the fundamental tasks in computer vision. It is used almost everywhere: human-computer interaction, video surveillance, medical treatments, robotics, smart cars, etc. The goal of object tracking is to estimate the state of the selected object in the subsequent frames [4]. The object being tracked is usually marked using a rectangle to indicate its location in the starting frame [5].

Although object tracking has been studied for several decades and considerable progress has been made in recent years, it remains a challenging problem. Numerous factors affect the performance of a tracking algorithm, including illumination variation, occlusion, and background clutters, and there exists no single approach that successfully handles all scenarios [4].

OpenCV object tracking is a popular method because OpenCV has so many algorithms built-in that are specifically optimized for the needs and objectives of object or motion tracking. The OpenCV library includes eight algorithms for object tracking, which is available through OpenCV tracking API. In the literature, we provided some information about the available algorithms in the OpenCV library with their publication years and reference to research papers detailing their implementation [5]. The OpenCV is a well-known library, which integrates necessary structures and tools for computer vision algorithms; in addition, it integrates a large set of different pre-implemented algorithms solving different parts of the object tracking problems. It is distinguished by its versatility and simplicity of use[6].

In general, tracking an object in the video involves steps such as: a) choosing the tracker, b) selecting the object (target) from the starting frame with the bounding box, c) initializing the tracker with information about the frame and bounding box, and d) reading the remaining frames and finding the new bounding box of the object. The last step is usually implemented in the loop[5].

The main objective of this work is to track a single object by pre-implemented tracking algorithms available in the OpenCV library. We focused on tracking algorithms based on the correlation filter approach available in OpenCV, such as the MOSSE, KCF, and CSRT trackers. We focused on single object tracking, in which an object is being tracked even if the environment consists of multiple objects. We have provided results on the object tracking dataset by experimenting with the three trackers we selected from the OpenCV library. The experimental results obtained are expressed on two levels: quantitative (center location error and overlap rate) and qualitative (visual).

We have chosen to organize our study around three main chapters as follows:

- **The first chapter**, we will begin with a brief definition of object tracking and its field of applications and then see the challenges it faces. Next, we describe the representation of the object's shape and appearance using various features. Finally, we presented the state of the art of object tracking methods.
- **The second chapter,** we will first present an overview of the OpenCV library. then, we talked about object tracking with OpenCV and the solutions it offers, following that, we are going to provide information about the available algorithms in the OpenCV library. Finally, we'll go through the principles of the correlation filter tracking approach and how its algorithms have developed over time, then we will see the principle of the correlation filter tracking algorithms in OpenCV.
- **The third chapter,** we will implement the tracking algorithms the MOSSE, KCF, and CSRT algorithms in OpenCV. Then we will present the results obtained and will do a comparative study between the three trackers.

We will end this work with a general conclusion and the perspectives.

Chapter 1

General information on object tracking

1.1 Introduction

Object tracking is an important task within the field of computer vision. The proliferation of high-powered computers, the availability of high-quality and inexpensive video cameras, and the increasing need for automated video analysis have generated a great deal of interest in object tracking algorithms[7]. The goal of tracking is to estimate the states of the target in the subsequent frames.[8]

In this chapter, we will go over the principles of object tracking and its applications, as well as the challenges that tracking algorithms encounter. We'll also look at the two different types of tracking algorithms. Next, we will describe the representation of the object's shape and appearance using the different features presented. Finally, we will present the state of the art of object tracking methods, which are classified into several categories: Tracking by detection, Tracking by correspondence, Tracking by correlation filter ...etc.

1.2 Object Tracking

Rapid developments of artificial intelligence and computer vision have been widely visible in various fields. Computer vision refers to the use of cameras and computers instead of human eyes to visually recognize, track, and measure targets. First, image processing is performed so that the processed image is more suitable for human eye observation or instrument detection. Then, the process of visual object tracking is to track the target state in subsequent video sequence frames with the presented initial position and size of the target. Currently, object tracking is widely used in transportation hub monitoring, medical imaging, human–computer interaction, and other related fields [9].

In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene [7]. In other words, object tracking is the estimation of the location of the object in each of the frames of a video sequence and initially detected on the first frame. The localization process is based on the recognition of the object of interest from a set of visual characteristics such as color, shape, speed, etc. [10]

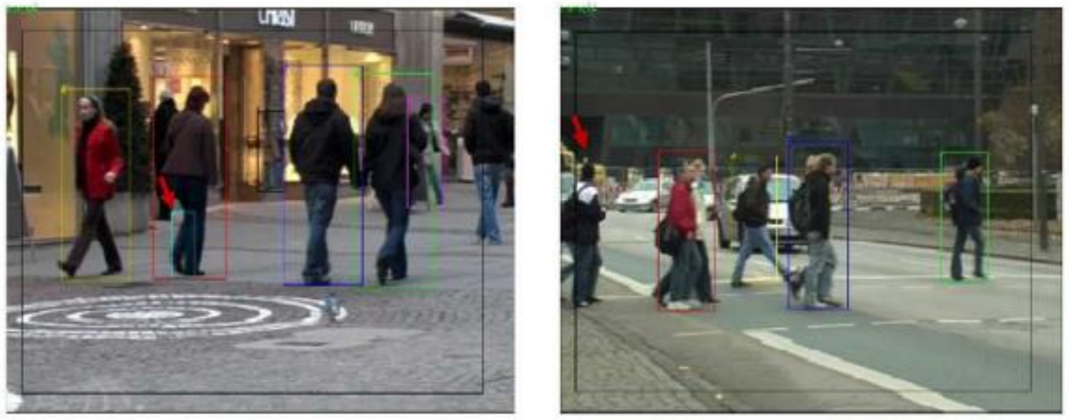


Figure 1.1: Object Tracking Examples.

Many object tracking methods have been proposed in the literature and the difference between these methods lies partly in the choice of object representation and shape, the features (components) of the image used, the nature of estimated motion, etc. This choice depends on the application as well as the processed video.[7]

An object tracking method aims to estimate object in each image of the sequence [11]. The object being tracked is usually marked using a rectangle to indicate its location in the starting frame (the starting frame does not have to be the first frame in a video sequence). When there are no changes in the environment, object tracking is not overly complex, but this is rarely the case [5].

1.2.1 Object Tacking Applications

Tracking objects in video sequences has sparked great interest over the past decade due to the variety of its application areas (see figure 1.2), such as: [12]

- Video surveillance (detection, tracking, recognition of the behavior of people).
- Human-computer interactions.
- Robotics (following obstacles during an avoidance phase).
- Traffic management and analysis (car tracking).
- Military (tracking targets and missile guidance).
- Medical imaging.

Figure 1.2: Example of object tracking applications

1.2.2 The Challenges in Object Tracking

Although object tracking has been studied for several decades and considerable progress has been made in recent years, it remains a challenging problem. Numerous factors affect the performance of a tracking algorithm, such as[8]:

- Illumination variation.
- Occlusion.
- Background clutters.
- Scale variation.
- Fast Motion.
- Motion Blur.
- Out-of-Plane Rotation.
- In-Plane Rotation.
- object deformation.

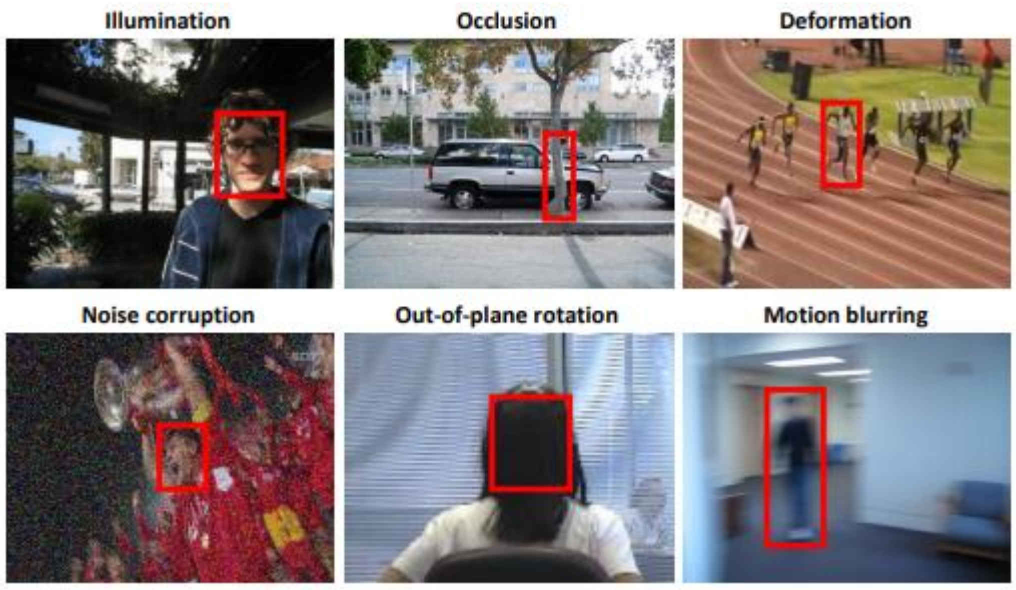


Figure 1.3: Some challenges in object tracking

1.2.3 Types of Object Tracking Algorithms

Object tracking tasks can be classified based on how many objects are being tracked in a sequence into[13]:

a) **Single Object Tracking (SOT):**

A Single Object Tracking (SOT) algorithm tracks only a single object in a video sequence, and it is successful if it tracks an object even if the environment consists of multiple objects. The process of (SOT) is selecting a region of interest (in the initial frame of a video) and tracking the position (i.e. coordinates) of the object in the upcoming frames of the video. In this work, we will be covering some of the algorithms used for single object tracking.

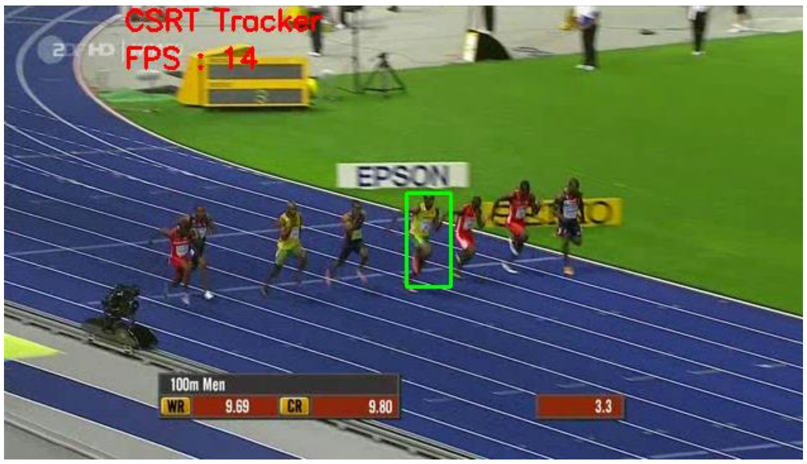


Figure 1.4: Single Object Tracking Example

b) Multiple Object Tracking (MOT):

Multiple object tracking is the task of tracking more than one object in the video. In this case, the algorithm assigns a unique variable to each of the objects that are detected in the video frame. Subsequently, it identifies and tracks all these multiple objects in consecutive/upcoming frames of the video.

Since a video may have a large number of objects, or the video itself may be unclear, and there can be ambiguity in direction of the object's motion Multiple Object Tracking is a difficult task and it thus relies on single frame object detection. Figure 1.5 illustrates the MOT process.

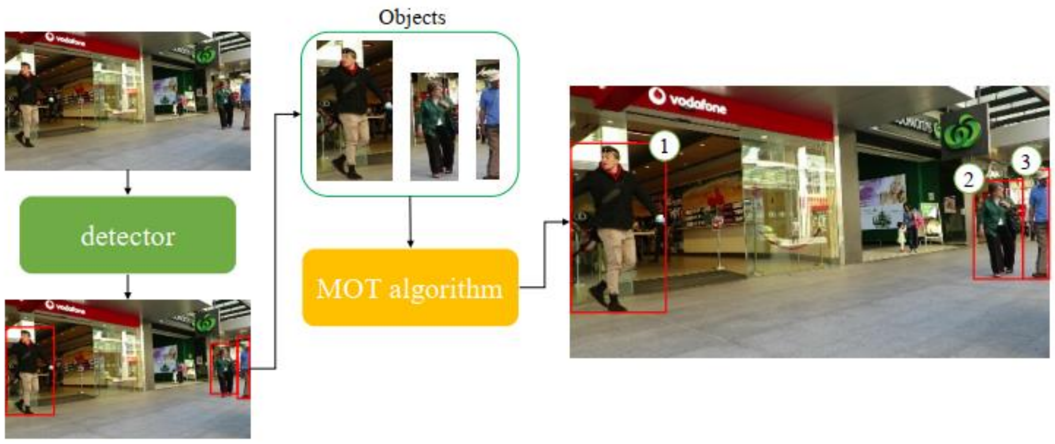


Figure 1.5: Multiple Object Tracking Process

1.3 Object Representation

Like other tasks in computer vision, visual representation plays a fundamental role in object tracking. Most object tracking methods are based on an object's appearance. The use of these methods requires a relevant representation of the object with reliable primitives to describe its content. Objects can be represented in many ways in terms of their shapes and appearances. Some approaches use only the shape of the object to represent it, but some also combine shape and appearance. The choice of the representation of an object strongly depends on the domain of application [10]. Yilmaz et al [1] were the first to propose a classification of object representation. In this section, we will use this classification.

1.3.1 Representation of the shape of an object

There are many representations based on the shape of an object: a set of points, a geometric shape (e.g. a rectangle, an ellipse), a contour, a silhouette [14].

- **Points:** The object is represented by a point (center of the object) (Figure 1.6 (a)) or by a set of points (Figure 1.6 (b)). In general, the point representation is suitable for tracking objects that occupy small regions in an image.[7]
- **Primitive geometric shapes:** The object is represented by a simple geometric shape such as a rectangle (Figure 1.6 (c)), an ellipse (Figure 1.6 (d)), etc. allowing a description of the dimension of the object. This representation is particularly suitable for tracking rigid objects (vehicles, etc.) but can also be used for non-rigid objects.[10]
- **Silhouette and contour:** Contour representation define the boundary of an object (Figure 1.6 (g), (h)). The region inside the contour is called the silhouette of the object (see Figure 1.6 (i)). Silhouette and contour representations are suitable for tracking complex nonrigid shapes.[7]
- **Articulated shape models:** Articulated objects are composed of body parts that are held together with joints. For example, the human body is an articulated object with the torso, legs, hands, head, and feet connected by joints. The relationship between the parts is governed by kinematic motion models, for example, joint angle, etc. To represent an articulated object, one can model the constituent parts using cylinders or ellipses as shown in Figure 1.6 (e).[7]
- **Skeletal models:** The object skeleton can be extracted by applying medial axis transform to the object silhouette. This model is commonly used as a shape representation for recognizing objects. Skeleton representation can be used to model both articulated and rigid objects (see Figure 1.6 (f)).[7]

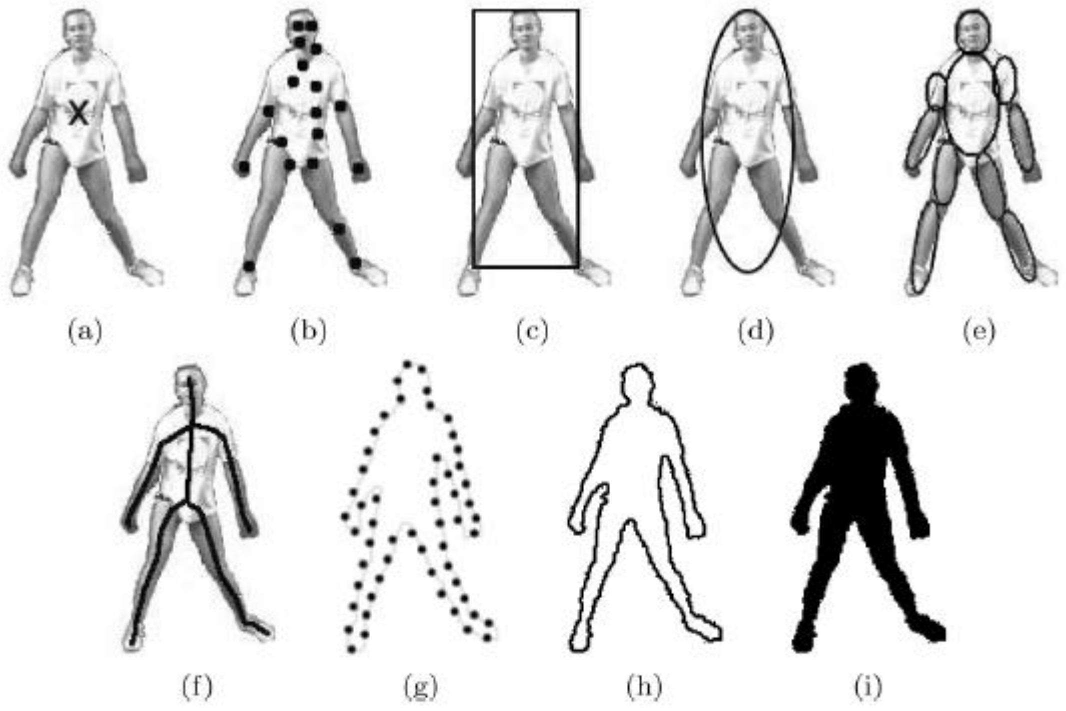


Figure 1.6: Examples of Object representations. (a): Centroid, (b): multiple points, (c): rectangular, (d): elliptical, (e): multiple blocks, (f): object skeleton, (g): complete object contour, (h): object contour, (i): object silhouette.

1.3.2 Representation of the appearance of an object

There are several ways to represent the appearance features of objects. Note that shape representations can also be combined with appearance representations for tracking. Some common appearance representations in the context of object tracking are [7]:

- **Probability densities of object appearance:** The probability density of object appearance features (color, texture) can be computed from the image regions specified by the shape models (interior region of an ellipse or a contour). The probability density function of an object can be parametric, such as Gaussian or a mixture of Gaussians. Similarly, non-parametric kernel based and histograms are also used.
- **Templates:** Templates are formed using simple geometric shapes or silhouettes. An advantage of a template is that it carries both spatial and appearance information. Templates, however, only encode the object appearance generated from a single view. Thus, they are only suitable for tracking objects whose poses do not vary considerably during tracking.

- **Active appearance models:** Active appearance models are generated by simultaneously modeling the object's shape and appearance. In general, the object's shape is defined by a set of landmarks. Similar to the contour-based representation, the landmarks can reside on the object boundary or they can reside inside the object region. For each landmark, an appearance vector is stored which is in the form of color, texture, or gradient magnitude. Active appearance models require a training phase where both the shape and its associated appearance are learned from a set of samples using, for instance, the principal component analysis.

- **Multiview appearance models:** These models encode different views of an object. One approach to represent the different object views is to generate a subspace from the given views. Subspace approaches, for example, Principal Component Analysis (PCA) and Independent Component Analysis (ICA), have been used for both shape and appearance representation. Another approach to modeling appearance is to train a set of classifiers to represent different views of the object (e.g. Support Vector Machines (SVM)) [7] [15].

1.4 Visual Features for Object Tracking

Selecting the right features plays a critical role in tracking. Generally, features that better distinguish between multiple objects and between object and background are best for object tracking. These features aim to describe the object's visual properties in the image. Some methods are based on a single type of feature, while others use a weighted combination of multiple types of features to improve performance. The details of common visual features are as follows [10].

- **Color:** The apparent color of an object is influenced primarily by two physical factors: 1) the spectral power distribution of the illuminant, and 2) the surface reflectance properties of the object. In image acquisition, the RGB (red, green, blue) color space is usually used to represent color. However, the RGB space is not perceptually uniform, that is, the difference between the colors in the RGB space does not correspond to the color differences perceived by humans. Instead, YUV and LAB are perceptually uniform, while HSV (Hue, Saturation, Value) is an approximately uniform color space. However, these color spaces are sensitive to noise. In summary, there is no last word on which color space is more efficient, therefore a variety of color spaces have been used in tracking [16].

- **Gradient:** Object boundaries usually generate strong changes in image intensities. Edge gradient identifies these changes. An important property of edges is that they are less sensitive to illumination changes as compared to color features. Algorithms that track the boundary of the objects usually use edges as the representative feature. Because of its simplicity and accuracy, the most popular edge detection approach is the Canny Edge detector [16].
- **Texture:** Texture is a measure of the intensity variation of a surface which quantifies properties such as smoothness and regularity. Compared to color, texture requires a processing step to generate the descriptors. There are various texture descriptors including gray level co-occurrence matrices, Law's texture measures (twenty-five 2D filters generated from five 1D filters corresponding to level, edge, spot, wave, and ripple), wavelets, Gabor filters, and steerable pyramids. A detailed analysis of texture features can be found in[17]. the texture features are less sensitive to illumination changes as compared to color. [16]
- **Optical Flow:** Optical flow is a dense field of displacement vectors that defines the translation of each pixel in a region. It is computed using a brightness constraint, which assumes "brightness constancy" of corresponding pixels in consecutive frames and is usually computed using the image derivatives. Optical flow is commonly used as a feature in motion-based segmentation and tracking applications [16]. A comparison of the popular optical flow techniques can be found in [18].

1.5 Object tracking methods

Many object tracking approaches have been proposed and the difference between these methods lies partly in the choice of object representation and shape, the image features used, the nature of the estimated motion, etc. This choice depends on the application as well as the processed video. There is more than one possible categorization of tracking algorithms in the literature. Yilmaz et al, [1] propose a classification of tracking methods according to the object representation used: point tracking, kernel tracking, and silhouette tracking. Recently, another classification for tracking algorithms based on the appearance model used. In [19] and [20], the authors distinguish between two categories: generative and discriminative methods. Generative methods focus on modeling the appearance of the object, which can vary in a different frame. Discriminative methods distinguish the object from the background, transforming the tracking problem into a binary classification problem [10]. Li et al [20] provide a very detailed description of all the existing appearance models in tracking and discuss their composition (visual representation and statistical modeling of appearance) [11]. In this section, we take the classification of tracking algorithms (tracking by detection, tracking by correspondence, tracking by correlation filter, and tracking by deep learning), The majority of the methods we'll see in this classification of tracking algorithms are built-in to the OpenCV library. which presented in the following figure.

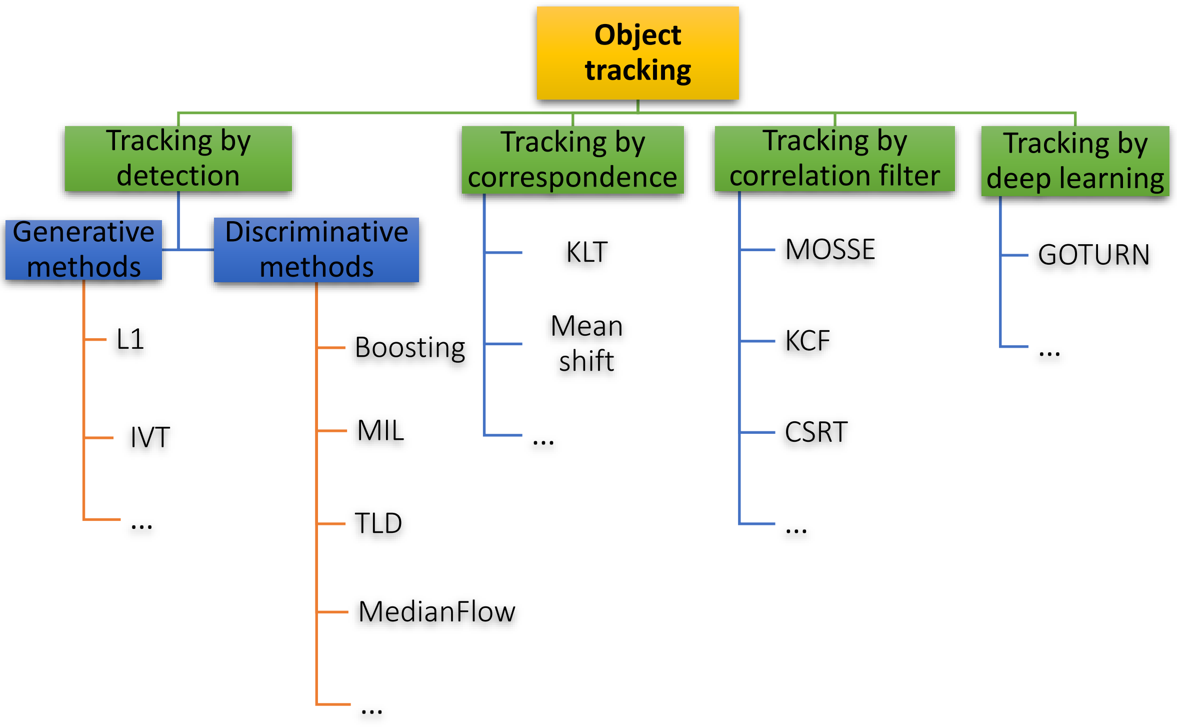


Figure 1.7: Taxonomy of object tracking methods.

1.5.1 Tracking by detection

During the past several years, tracking by detection has become one of the most successful paradigms for visual object tracking and has achieved state-of-the-art performance. The two main components in the tracking-by-detection approach are visual representation and statistical modeling. In this part, we will focus on the most representative papers related to statistical modeling, which can be grouped into two categories: generative and discriminative models [21].

1.5.1.1 Generative Method

Tracking with generative modeling typically focuses on learning a model to represent the target object, and then uses it to find the most similar region in the future frames [21]. In general, this category of method does not require a large data set for training.[22]

- **Tracker L1 :** Mei et Ling. [23] proposed a robust tracking method, this method treats object tracking as an approximation problem by acrimonious and introduces the trivial model to approximate noise and occultation. During tracking, target candidates are represented as a sparse linear combination of model sets including target models that are obtained from previous frames and trivial models. The L1 tracker requires high computational resources due to many L1 minimization calculations.
- **Tracker IVT:** Ross et al. [24] proposed a tracking algorithm that uses an incremental subspace model to describe the target object to adapt to appearance changes. It carries out the incremental learning of an object representation subspace (PCA) and adapts the model by integrating the new appearance of the object with a forgetting factor on the past appearances of the object. This method is not very robust, especially when the location of the object is imprecise.

1.5.1.2 Discriminative Method

Despite the success, generative modeling usually faces difficulties to describe the target object without considering background information, especially when the appearance of target object changes dramatically and/or the background is cluttered. On the contrary, instead of trying to build a complete model to represent the target object, discriminative modeling treats object tracking as a classification problem, in order to distinguish the target object from the background. Therefore, it is usually more robust to complex scenarios by explicitly modeling background as negative training samples. Trackers based on discriminative modeling have evolved rapidly and dominated almost all datasets in recent years [21].

- **Boosting Tracker:** Grabner et al. [25] applied a similar online boosting framework for real-time object tracking [21], This method is based on the online version of the AdaBoost algorithm, the algorithm increases the weights of incorrectly classified objects, which allows a weak classifier to “focus” on their detection. Since the classifier is trained “online”, the user sets the frame in which the tracking object is located. This object is initially treated as a positive result of detection, and objects around it are treated as the background. Receiving a new image frame, the classifier scores the surrounding detection pixels from the previous frame and the new position of the object will be the area where the score has the maximum value.[25]
- **MIL Tracker:** Babenko et al. [26] proposed to apply multiple instance learning (MIL) in visual object tracking, in order to allow the classifier to select from a number of potential positive samples according to its current state. MIL tracker treats the training samples as “bags”. A bag is considered as positive if it contains at least one positive instance, otherwise the bag is set to negative [21].
- **TLD:** Kalal et al. [27] proposed a robust visual tracking algorithm. this algorithm decomposed long-term tracking task into three sub-tasks: tracking, learning and detection, and their corresponding components are designed to form a general tracker, called TLD. The tracking component estimates the object motion and follows the object continually in order to produce smooth trajectories, but it also accumulates errors continually and fails to track if the target is invisible. The detection component localizes the object in all its appearances that have been observed so far, and reinitializes tracker when it fails. The learning procedure estimates the quality of the results and updated it with only high reliable results [21].
- **MEDIANFLOW Tracker:** This algorithm is based on the Lucas-Kanade method. The algorithm tracks the movement of the object in the forward and backward directions in time and estimates the error of these trajectories, which allows the tracker to predict the further position of the object in real-time.[28]

1.5.2 Tracking by correspondence

Matching the representation of a target object between two consecutive frames is a natural way to estimate its motion and track it over time. This was a dominant tracking approach in the early days, due to its fairly good performance, simple structure, and low computational requirement [21].

- **KLT:** As a more effective template-matching approach, Lucas and Kanade [29], proposed The KLT (Kanade-Lucas-Tomasi) tracker. The KLT tracker finds transformed affine correspondences between two successive frames using Spatio-temporal derivatives. The target's new location is determined by matching its position in the previous frame to the location in the current frame using the estimated affine transformation.
- **Mean shift:** Several approaches have emerged to solve the lack of robustness to track non-rigid objects based on the mean shift algorithm proposed by [Fukunaga et al,1975] which was originally proposed for data clustering. Comaniciu et al [30], designed mean-shift trackers to perform matching with color histograms, which is invariant to changes in the shape of targets. In every new frame, the mean shift algorithm is used to determine the location of the target by maximizing a similarity metric. This tracker, however, can be confused by regions with a similar color distribution, due to the lack of spatial information.

1.5.3 Tracking by Correlation Filter

In recent years, correlation filter-based trackers have received much attention due to their simple structure, state-of-the-art performance and computational efficiency. As a basic operation in digital image processing, a correlation filter is used to find locations in an image that are similar to a pre-defined template. Ideally, a correlation filter produces high responses for a pre-defined template, while generating low responses for background [21].

- **MOSSE Tracker:** Bolme et al. [31] proposed the minimum output sum of squared error (MOSSE) filter for visual tracking on grayscale images. This algorithm is based on the calculation of adaptive correlations in Fourier space. The filter minimizes the sum of squared errors between the actual correlation output and the predicted correlation output. MOSSE filter-based tracking is computationally efficient with speeds up to several hundred frames per second and robust to variations in illumination, scale, pose, and non-rigid deformations.

- **KCF Tracker :** Henriques et al. [32] proposed KFC (Kernelized Correlation Filters). This tracker builds on the ideas presented in the previous two trackers (BOOSTING and MIL). This tracker utilizes the fact that the multiple positive samples used in the MIL tracker have large overlapping regions. This overlapping data leads to some nice mathematical properties that are exploited by this tracker to make tracking faster and more accurate at the same time [33].
- **CSRT Tracker:** CSRT is the implementation of the CSR-DCF (Channel and Spatial Reliability of Discriminative Correlation Filter). This algorithm uses spatial reliability maps for adjusting the filter support to the part of the selected region from the frame for tracking, which gives an ability to increase the search area and track non-rectangular objects. Reliability indices reflect the quality of the studied filters by channel and are used as weights for localization. Thus, using HoGs and Colornames as feature sets, the algorithm performs relatively well [34] [35].

1.5.4 Tracking by deep learning

In recent years approaches based on Deep Learning, in particular, the convolutional neural network (CNN), have achieved great empirical success and have dominated many computer vision problems, the application of deep learning to the problem of Visual object tracking requires extra effort, due to the lack of proper training data, and the target object changing during the video sequence [21].

GOTURN Tracker: GOTURN, short for (Generic Object Tracking Using Regression Networks). It is a Deep Learning-based tracking algorithm, based on Convolutional Neural Network (CNN) [36]. From OpenCV documentation, we know it is “robust to viewpoint changes, lighting changes, and deformations”. But it does not handle occlusion very well [33].

1.6 Conclusion

An important area of computer vision is real-time object tracking, which is now widely used. In this chapter, we covered a general idea about object tracking and its methods. we first discussed the theory of object tracking and its applications, as well as the challenges that affect the tracking algorithms performance, we saw also the two types of tracking algorithms. Then we described the representation of the object's shape and appearance using the different features presented. Finally, we presented the state of the art of object tracking methods, which are classified into several categories: Tracking by detection, tracking by correspondence, Tracking by correlation filter, ... etc. The majority of the methods we saw in the classification of tracking algorithms are algorithms built-in on the OpenCV library.

Chapter 2

Object tracking by OpenCV with MOSSE, KCF, CSRT

2.1 Introduction

Locating an item in consecutive frames of a video is known as object tracking. It is implemented by estimating the state of the concerned object present in the scene from previous information. Since the object has been tracked till the present frame, it's known how it has been moving. More simply, the parameters of the model are known. A motion model tells the speed and direction of motion of the object from previous frames. Algorithms that track objects using this motion model are known as object tracking algorithms [37]. There is a multitude of algorithms that can be used for the same purpose. OpenCV offers a number of pre-built algorithms developed explicitly for the purpose of object tracking.

In this chapter, we will first present an overview of the OpenCV library. then, we'll look into the three main components that make up OpenCV trackers. In section 4, we provide information about the available algorithms in the OpenCV library. Finally, in section 5 we will briefly go through the correlation filter tracking approach basics and how its algorithms have developed over time, and then we will focus on the three correlation filter tracking algorithms in OpenCV.

2.2 OpenCV library

OpenCV was started at Intel in the year 1999 by **Gary Bradsky.** The first release came a little later in the year 2000. Open Source Computer Vision (OpenCV) library is an open source cross-platform computer vision and machine learning software library. Some of its initial goals still hold today, such as providing optimized code for basic computer vision infrastructure, disseminating knowledge to build applications faster, and providing portable, performance-optimized code. The library is licensed with open-source BSD license, and can be used for academic and commercial applications [5].

Today OpenCV has around 2,500 optimized algorithms [38], which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can used to detect and recognize human faces, identify various objects, classify human actions in video, track moving objects, extract 3D models of objects, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

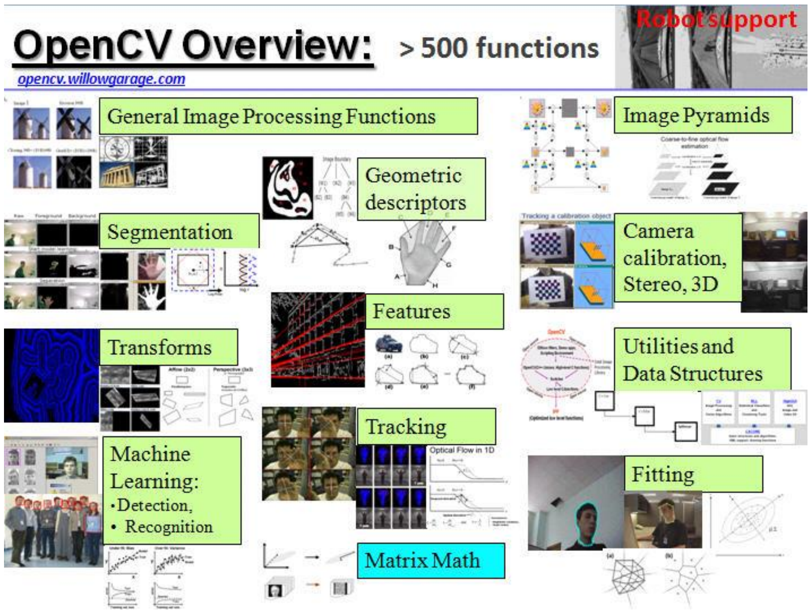


Figure 2.1: Different types of algorithms in OpenCV

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many start-ups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan[39].

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers**.** The latest stable OpenCV version release was published in December 2021 as 4.5.5 version [39].

2.3 Object tracking with OpenCV

Most modern solutions of the object tracking problem assume the presence of a pre-trained classifier that allows you to accurately determine the object you track, whether it is a car, person, animal, etc. These classifiers are, as a rule, trained on tens to hundreds of thousands of images, which makes it possible to study the patterns of the selected classes and subsequently to detect the object. But, what if the user can't find a suitable classifier or train their own?

In this case, OpenCV object tracking provides solutions that use the "online" or "on the fly" training [34]. The term online refers to algorithms that are trained using very few examples at run time [8], in contrast to an offline classifier, which may need thousands of examples to train [33].

In general, tracking an object in the video involves steps such as [40]:

- Choosing the tracker,
- Selecting the object (target) from the starting frame with the bounding box,
- Initializing the tracker with information about the frame and bounding box,
- Reading the remaining frames and finding the new bounding box of the object. The last step is usually implemented in the loop.

An OpenCV tracker consists of three main components, which also coincide with the components in a typical tracking algorithm [5]:

2.3.1 Tracker Feature Set:

The tracker feature set is a model of the visual appearance of the target object, and it is used to represent objects of interest. In OpenCV, possible features can be extracted using HAAR, HOG, LBP, Feature2D, etc.

2.3.2 Tracker Sampler Algorithm :

The mechanism for matching model parts to image regions at each frame through computes the patches over the frame based on the last target location.

2.3.3 Tracker Model:

The mechanism for continuously relearning or updating models of targets that change their appearance over time. The internal representation of the target. It stores all state candidates and computes the trajectory. Tracker Feature Set and Tracker Sampler Algorithm are the visual representation of the target, while the Tracker Model represents the statistical model.

2.4 Object tracking algorithms in OpenCV

OpenCV object tracking is a popular method because OpenCV has so many algorithms built-in that are specifically optimized for the needs and objectives of object or motion tracking.

The OpenCV library includes eight algorithms for object tracking the BOOSTING, MIL, KCF, CSRT, MedianFlow, TLD, MOSSE, and GOTURN algorithms, which are available through OpenCV tracking API. Each of these trackers is best for different goals. For example, CSRT is best when the user requires a higher object tracking accuracy and can tolerate slower FPS throughput, and the selection of an OpenCV object tracking algorithm depends on the advantages and disadvantages of that specific tracker [5] [37].

Table 2.1 provides some information about the available algorithms in the OpenCV library with their publication years and reference to research papers detailing their implementation.

Table 2.1: OpenCV single object trackers sorted by the year of their publication. Google Scholar Citations are accessed on May 15th 2022.

N°	**Tracker Full Name** (**Abbreviation**)	**Publication Title and Reference**	**Publication Year** (**Google Scholar Citations**)
1.	**Boosting**	Real-time tracking via on-line boosting [25]	2006 (1432)
2.	Multiple Instance Learning (**MIL)**	Visual tracking with online multiple instance learning [26]	2009 (2095)
3.	**MedianFlow**	Forward-backward error: Automatic detection of tracking failures [28]	2010 (802)
4.	Minimum Output Sum of Squared Error (**MOSSE**)	Visual object tracking using adaptive correlation filters [31]	2010 (1839)
5.	Tracking Learning Detection (**TLD**)	Tracking-learning-detection [27]	2011 (3275)
6.	Kernelized Correlation Filter (**KCF**)	High-speed tracking with kernelized correlation filters [32]	2014 (3131)
7.	**GOTURN** (Generic Object Tracking Using Regression Networks)	Learning to track at 100 fps with deep regression networks [36]	2016 (648)
8.	**CSRT** (Channel and Spatial Reliability Tracker)	Discriminative Correlation Filter with Channel and Spatial Reliability [35]	2017 (444)

2.5 Correlation filter tracking algorithms in OpenCV

In 2010, the correlation filter method was used in object tracking for the first time [9]. In recent years, correlation filter-based trackers have received much attention due to their simple structure, state-of-the-art performance, and computational efficiency [21]. In OpenCV, we have the MOSSE, KCF, and CSRT trackers based on the correlation filter method. In this section, we will briefly go through the correlation filter tracking approach basics and how its algorithms have developed over time, and then we will focus on the three correlation filter tracking algorithms that we have in OpenCV.

2.5.1 Correlation filter tracking

As a basic operation in digital image processing, a correlation filter is used to find locations in an image that are similar to a pre-defined template. Ideally, a correlation filter produces high responses for a pre-defined template, while generating low responses for background [21].

The framework of a typical correlation filter-based tracking method (Figure 2.3) can be summarized as follows. In the first frame, an initial correlation filter is trained based on the ground truth bounding box. For each following frame, various local features are extracted and filtered by a cosine window in order to smooth the boundary effects. Subsequently, a response map is generated efficiently with a fast Fourier transform (FFT). The position with the maximum value in this map is predicted as the new location of the target object. Finally, the new location of the object is used to update the correlation filter [21] [31].

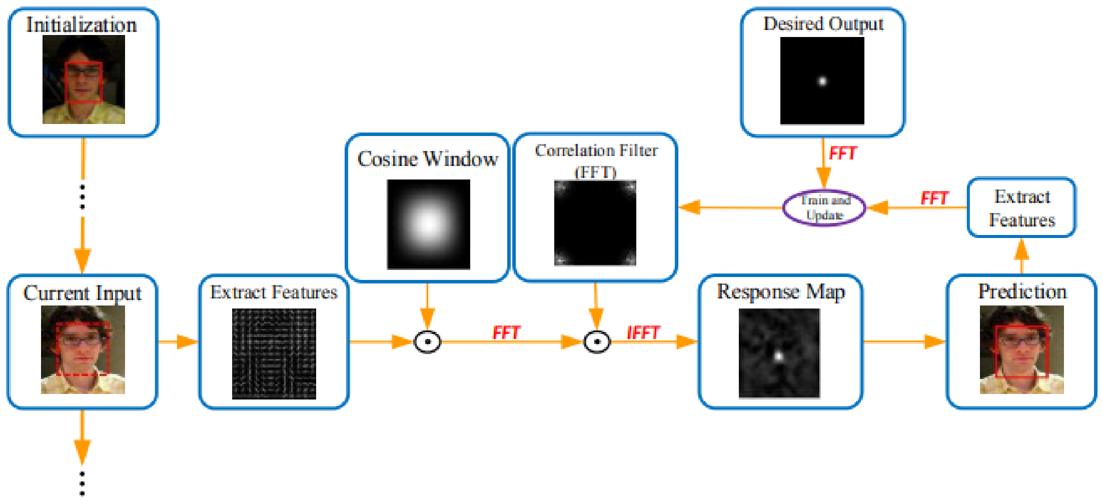


Figure 2.2:A general flow chart for typical correlation filter-based tracking methods.

After nearly a decade of development, the correlation filter tracking algorithms now have matured. In this part, we will introduce the development of correlation filter algorithms. The specific development process is as follows [9].

By learning from gray images, the minimum output sum of squared error (MOSSE) [31] filter applies correlation filter to the tracking field for the first time. This filter is easy to calculate and can quickly track objects, but it does not guarantee to track accurately when the object's appearance changes. After that, Henriques et al. [43] proposed the circulant structure tracking with kernels (CSK) in 2012. Then, Danwelljan et al. [32] proposed that the Kernels correlation filter (KCF) further adjusts the channel characteristics to multi-channel features and introduces CN features for tracking in 2014. The CN feature improves the filter's discriminative ability. However, the adaptability of the filter to rotation, out-of-view and fast motion still needs to be improved. Subsequently, Danelljan et al. [44] proposed a discriminative scale space tracker (DSST) using the feature pyramid to solve the multi-scale variation problem and also proposed the improved fDSST algorithm [45]. With the rapid rise of deep learning, the C-COT algorithm [46] effectively represents spatial position information with shallow CNN features, which is a combination of correlation filtering and CNN. The algorithm won the VOT2016 competition. Similar to C-COT, the CSR-DCF algorithm [35] also applies CNN features to the correlation filtering algorithm, which improved the robustness of the algorithms.

2.5.2 MOSSE algorithm

The MOSSE algorithm [31] introduced the correlation filter technology into the visual tracking field. This kind of algorithm can adapt to the problems of occlusion and rotation and achieve an amazing tracking speed of 669 fps. The MOSSE filter is trained by the first frame and can have strong robust performance for illumination, scale, and posture variation. When the target is occluded, the algorithm can determine the status of object tracking and update the filter parameters according to the PSR value. When the object reappears, it can be tracked again [9].

In the MOSSE algorithm, to create a fast tracker, the fast Fourier transform (FFT) is used to calculate the correlation in the Fourier domain. First, calculating the 2D Fourier transform of the input image ($F = F(f)$) and filter ($H = F(h)$). The convolution theorem states that correlation is the element multiplication in the Fourier domain[31]. The symbol $\odot$ represents element-by-element multiplication, (*) indicates complex conjugate and the representation of correlation is as follows:

$$g = f \otimes h \tag{2.1}$$

where g, f and h represent response output, input image and filter template, respectively. It can be seen that we only need to determine the filter template h to get the response output. The fast Fourier transform (FFT) is used in Eq. (2.1). Therefore, the convolution operation becomes a point multiplication operation, which greatly reduces the amount of calculation. That is, the above formula becomes[9]:

$$F(g) = F(f \otimes h) = F(f) \cdot F(h)^* \tag{2.2}$$

Then, the above formula is abbreviated as follows: $G = F \cdot H^*$

And the next task to track is to find the filter template H*: $H^* = \frac{G}{F}$

In the process of actual tracking, we need to consider the influence of factors such as the appearance of the object. At the same time, considering the m images of the object as a reference can significantly improve the robustness of the filter template. The MOSSE model formula is as follows [9]:

$$\min_{H^*} = \sum_{i=1}^{m} |H^* F_i - G_i|^2 \tag{2.3}$$

after a series of transformations, a closed solution is obtained:

$$H^* = \frac{\sum_i G_i \cdot F_i^*}{\sum_i F_i \cdot F_i^*} \tag{2.4}$$

The algorithm tracks the object by correlating filters on the search window in the next frame. The new position of the object is represented by the maximum value of the associated output. Then performs an online update in the new location. The tracker update method uses the following formula:

$$H_i^* = \frac{A_i}{B_i} \tag{2.5}$$

$$A_i = \eta G_i \odot F_i^* + (1-\eta)A_{i-1} \tag{2.6a}$$

$$B_i = \eta F_i \odot F_i^* + B_{i-1} \tag{2.6b}$$

In equation (2.5), The cumulative values of A_i and B_i over a set of initial frames are used to initialize the filter.

Finally, the Peak to Sidelobe Ratio (PSR) value is used for failure detection:

$$\mathrm{PSR} = \frac{\text{peak} \; - \mu}{\sigma} \tag{2.7}$$

In the experiment, the PSR value between 20 and 60 is considered to be a good tracking effect. When the PSR value is lower than 7, it is judged as tracking failure and the template is not updated [9]. The MOSSE algorithm overall can adapt to small-scale variation (the figure 2.4 present the ability of MOSSE tracker to quickly adapt to scale and rotation changes), but it cannot adapt to large-scale variation. In addition, the MOSSE algorithm uses grayscale features that are not powerful enough and expressive in general. The sample sampling of the MOSSE algorithm is still a sparse sampling, and the training effect is general.

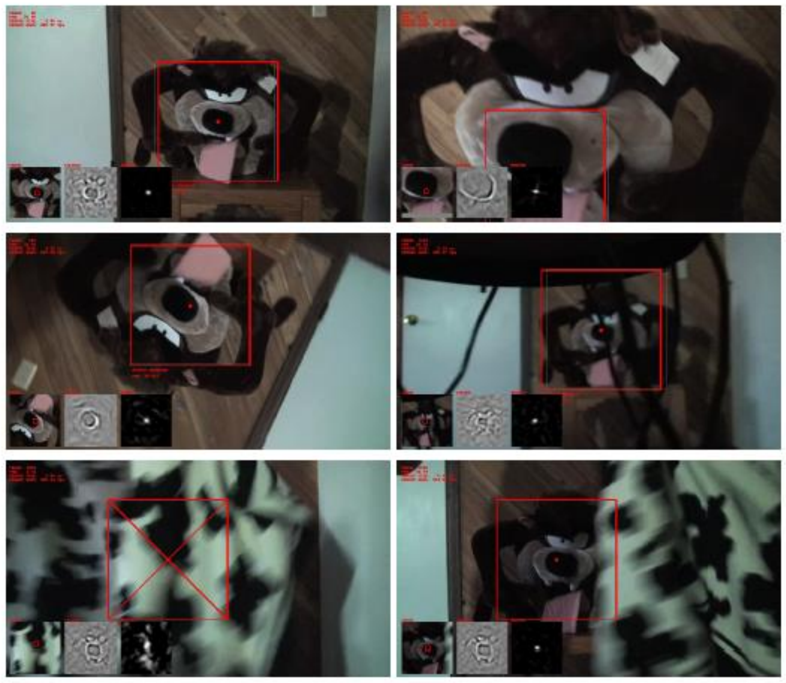


Figure 2.3: The results of the MOSSE filter based tracker on a challenging video sequence.

2.5.3 KCF algorithm

KCF stands for Kernelized Correlation Filter, it is a combination of techniques of two tracking algorithms (BOOSTING and MIL tracker). It is supposed to translate the bounding box (position of the object) using a circular shift. In simple words, the KCF tracker focuses on the direction of change in an image (could be motion, extension, or orientation) and tries to generate the probabilistic position of the object that is to be tracked[47] .

The KCF [32] is a classic of traditional discriminant method. This series of algorithms learn filters from a series of training samples. The KCF sample generation method uses the cyclic shift method. Assuming one-dimensional data as $x = [x_1, x_2, \dots, x_n]$ the cyclic shift of x is denoted as $Px = [x_n, x_1, \dots, x_{n-1}]$. All cyclic shift samples form a cyclic matrix are:

$$X = C(x) = \begin{bmatrix} x_1, x_2 & \cdots & x_n \\ \vdots & \ddots & \vdots \\ x_n, x_{n-1} & \cdots & x_1 \end{bmatrix} \qquad (2.8)$$

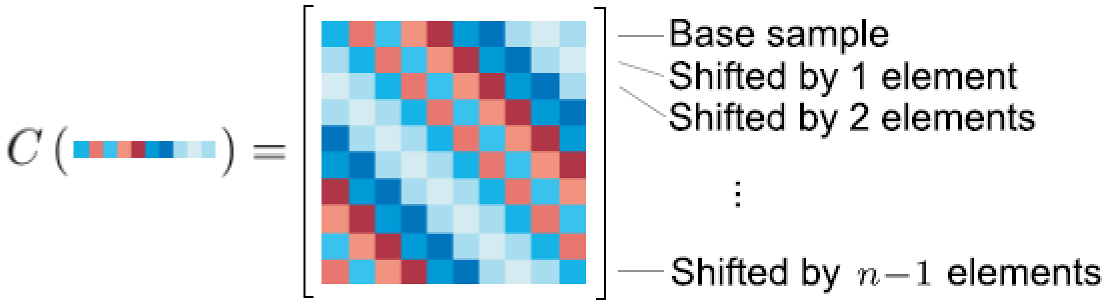


Figure 2. 4: Illustration of a circulant matrix.

That is, it uses ($M \times N$) image block x to train a filter $f(x) = \langle \omega, \emptyset_x \rangle$, which generates a training sample by performing a cyclic shift operation on *x*. The training samples include all cyclic shift forms *Pi*, where $i \in \{0, \dots, M-1\} \times \{0, \dots, N-1\}$. Each *Pi* generates a corresponding score *yi* (*yi* ∈ [0, 1]) which is generated by a Gaussian function based on the shift distance. Minimizing the regression error, the classifier is trained as:

$$w = \underset{w}{\operatorname{argmin}} \sum_i (\langle w, \phi(x) \rangle - y_i)^2 + \lambda \parallel w \parallel^2 \qquad (2.9)$$

Among them, $\phi(x)$ is the mapping of Fourier space. $\lambda \geq 0$ is the regularization parameter, which shows the simplicity of the model. The periodic hypothesis achieves effective training and detection by using fast Fourier transform. If the translation invariance of the kernel function is used, *a* can be quickly obtained as $\hat{\alpha} = \frac{\hat{y}}{\hat{k}^{xx}+\lambda}$ for the special nature of the circulant matrix. In the filtering conversion process, a $m \times n$ candidate image block z for the search space is evaluated by the following formula:

$$f(z) = \mathcal{F}^{-1}\left(\hat{k}^{xx} \odot \hat{\alpha}\right) \qquad \textbf{(2.10)}$$

Where $f(z)$ is the filter response of all cyclic matrices z, and the highest response is the object of the current frame.

The KCF algorithm generates a series of candidate samples by exploiting the properties of the cyclic matrix on the candidate window. It greatly improves the tracking speed compared to traditional window sampling. The problem is then converted to a fast operation in the frequency domain by Fourier transform. This turns the ridge regression problem in the time domain into a cross-correlation problem in the frequency domain. The KCF algorithm uses a multichannel HOG feature instead of a single-channel grayscale feature. Due to the use of cyclic shift, the KCF algorithm has a boundary effect problem. In addition, the search area is fixed in KCF, so it is easy to exceed the search range in fast motion[9].

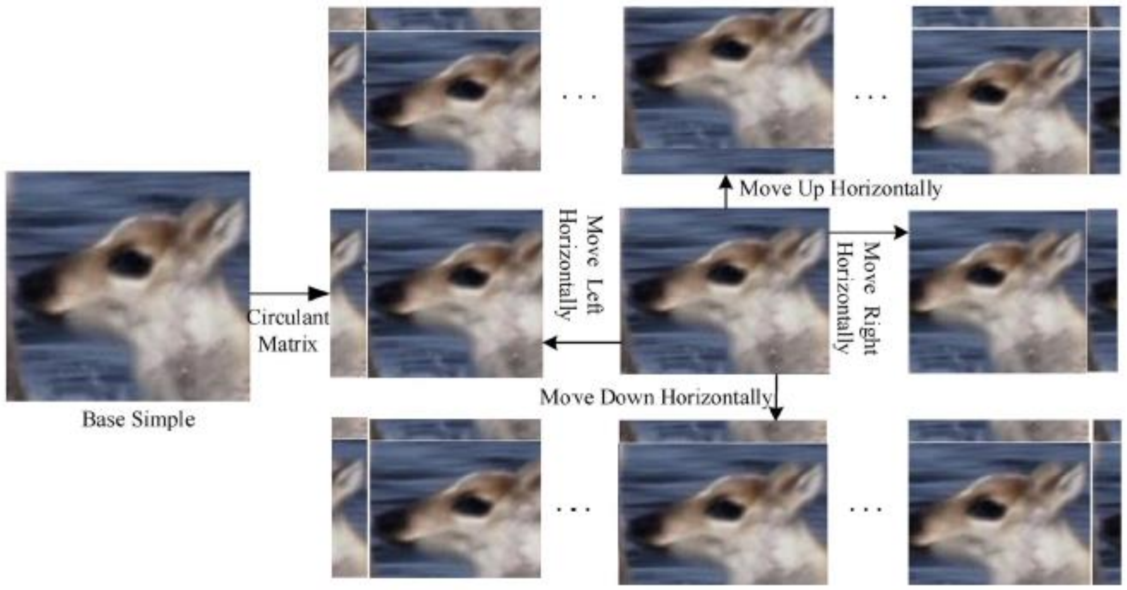


Figure 2.5: The principle and effect diagram of cyclic shift

2.5.4 CSRT algorithm

CSRT is the OpenCV implementation of the CSR-DCF (Channel and Spatial Reliability of Discriminative Correlation Filter), it is an advanced algorithm that accommodates changes like enlarging and non-rectangular objects. Essentially it uses HoG features along with SRM (spatial reliability maps) for object localization and tracking [35].

The spatial reliability map adapts the filter support to the part of the object suitable for tracking which overcomes both the problems of circular shift enabling an arbitrary search range and the limitations related to the rectangular shape assumption. The second novelty of CSR-DCF is the channel reliability. The reliability is estimated from the properties of the constrained least-squares solution. The channel reliability scores were used for weighting the per-channel filter responses in localization (see figure 2.6) [35]. An experimental comparison with the most recent state-of-the-art boundary-constraint method shows that there are significant advantages to using this method.

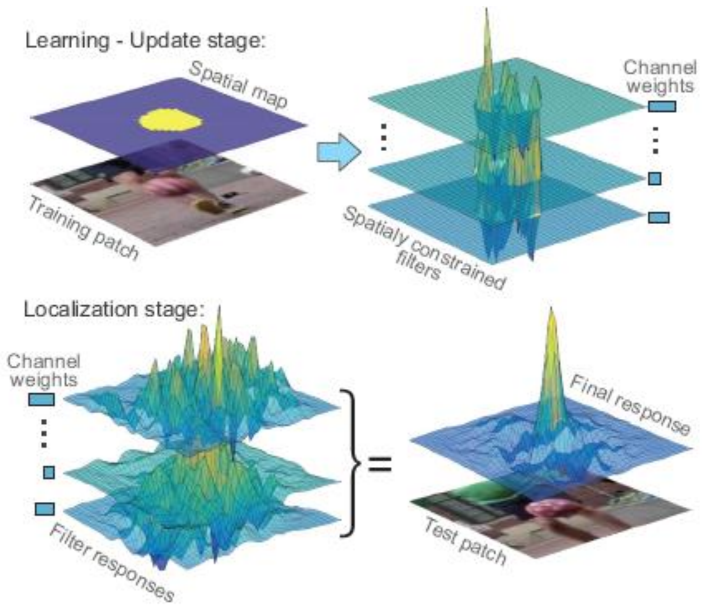


Figure 2.6: Overview of the CSR-DCF approach.

First, we construct a spatial reliability map that calculates the probability of occurrence from the target foreground/ background color model using Bayesian rules. The a priori probability is determined by the ratio of the size of the foreground/background histogram extracted regions (see figure 2.7). The method to construct the space reliability map is as follows [48]:

- Select the training patch tracking object boundary.
- Use Spatial priori as a unitary item of Markov random field optimization.

- Determine the object log-likelihood according to the foreground/background color model.
- Calculate the posterior probability of Markov random field regularization.
- Cover the training patch with the final binary reliability map.

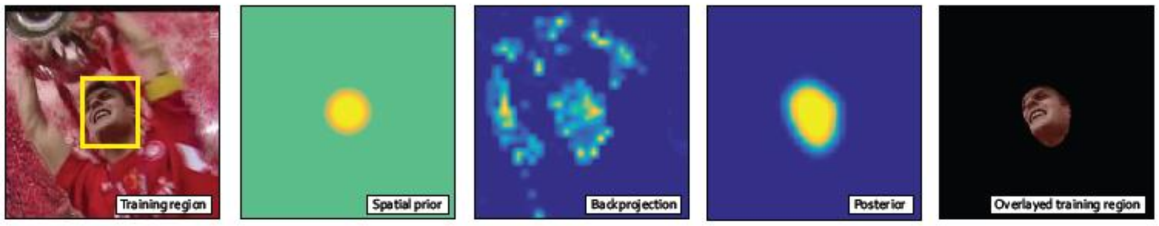


Figure 2. 7: Spatial reliability map construction from the training region.

2.5.4.1 Tracking with channel and spatial reliability

The localization and update steps of the channel and spatial reliability correlation filter tracker (CSR-DCF) proceed as follows [35].

- **Localization step:** Localize per-channel responses and the corresponding detection reliability values are computed and multiplied with the learning reliability measures from previous time-step $\mathbf{w}_{t-1}$ into channel reliability scores. The object is localized by summing the responses of the learned correlation filters $\mathbf{h}_{t-1}$ weighted by the estimated channel reliability scores. Scale is estimated by a single scale-space correlation filter from Danelljan et al [44] .
- **Update step:** Foreground/background histograms $\tilde{\mathbf{c}}$ are extracted at the estimated location and updated by an auto-regressive scheme with learning rate η_c. The foreground histogram is extracted by a standard Epanechnikov kernel within the estimated object bounding box and the background is extracted from the neighborhood twice the object size. The spatial reliability map is constructed, the optimal filters $\tilde{\mathbf{h}}$ are computed by optimizing the augmented Lagrangian and the per-channel learning reliability $\tilde{\mathbf{w}} = \left[\tilde{w}_1, \dots, \tilde{w}_{N_d}\right]^T$ is estimated from their responses. For temporal robustness, the filters and channel learning reliability weights are updated by an autoregressive model with learning rate η. The CSR-DCF tracking iteration is visualized in (Figure 2.8).

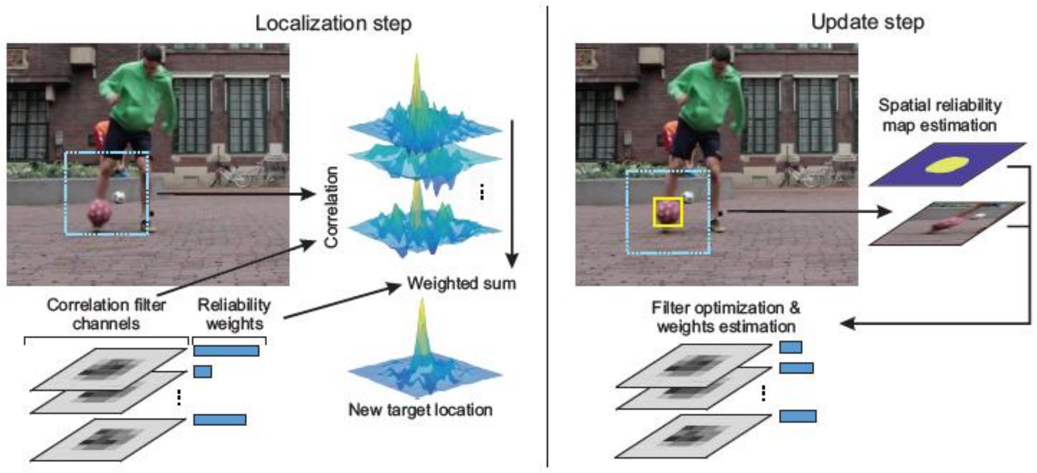

Localization step
Correlation
Weighted sum
Correlation filter channels
Reliability weights
New target location
Update step
Spatial reliability map estimation
Filter optimization & weights estimation

Figure 2.8: The CSR-DCF tracking iteration.

2.6 Conclusion

In this chapter, we have introduced the object trackers available in the OpenCV library. we started with a brief overview of the OpenCV library, then we saw the solutions provided by OpenCV to track objects using online training also the three main components of an OpenCV tracker, after that, we provided some information about the eight algorithms available in the OpenCV library with their reference detailing their implementation. In the last part we focused on the MOSSE, KCF, and CSRT trackers based on the correlation filter method, we introduced the basics of the correlation filter tracking approach and the development of its algorithms over time, then we went over how the three correlation filter trackers in OpenCV function. In the next chapter, we will see how to implement these three trackers in OpenCV.

Chapter 3

Simulation and results

3.1 Introduction

After taking a theoretical knowledge in the previous chapter of the basic principle of MOSSE, KCF, and CSRT tracking algorithms. in this chapter, we will implement these object tracking algorithms in video using OpenCV. The results obtained as well as their discussions will be presented. To validate this work, we tested the algorithms on different videos.

3.2 Hardware

In order to carry out this work project, a set of the following materials was made available to us:

- PC ASUS X541U Series with the following features:
    - Processor**:** Intel(R) Core (TM) i3-6006U CPU @ 2.00GHz.
    - RAM: 4.0 GB.
    - Hard disk: 128 GB SSD + 500 GB HDD.
    - laptop's in-built camera to build a real time object tracking.
    - Graphic Card: Intel(R) HD Graphics 520.
    - System type: 64-bit Operation System.
    - Windows edition: Windows 10 Pro.

3.3 Software

For this project, we used a set of development tools such as: The programming language Python 3.9, The computer vision library OpenCV 4.5.5, and PyCharm IDE.

3.3.1 Python

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991 [49]. the language is named after the BBC show "Monty Python's Flying Circus". Python 3.x is the current version and is under active development, the latest version of it is 3.10.4 was released in March 2022 [50].

Python is a popular and in-demand skill to learn. Python can be used for many areas such as: AI and machine learning, data analytics, Programming applications, Web development, and many other fields.

3.3.1.1 Python features [51]:

- **Python is open source.**
- **Python is simple and lovely:**

  Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#.
- **Python is portable:**

  Python scripts can be used on different operating systems such as: Windows, Linux, UNIX, Amigo, Mac OS, etc. You can move Python programs from one platform to another, and run it without any changes.
- **A high-level, interpreted language:**

  Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on. Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.
- **Large standard libraries to solve common tasks:**

  Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself.
- **Python supports other technologies:**

  It can support COM, .Net, etc objects.
- **Large standard libraries to solve common tasks**
- **Extensible and Embeddable**
- **Object-oriented**

However, there are few drawbacks with python:

- **Not Easy to Maintain:**

  Because Python is a dynamically typed language, the same thing can easily mean something different depending on the context.
- **Slow:**

  As a dynamically typed language, Python is slow because it is too flexible and the machine would need to do a lot of referencing to make sure what the definition of something is, and this slows Python performance down.

Here, in (figure 3.1) we can see how Python is the most in-demand programming language of 2022.

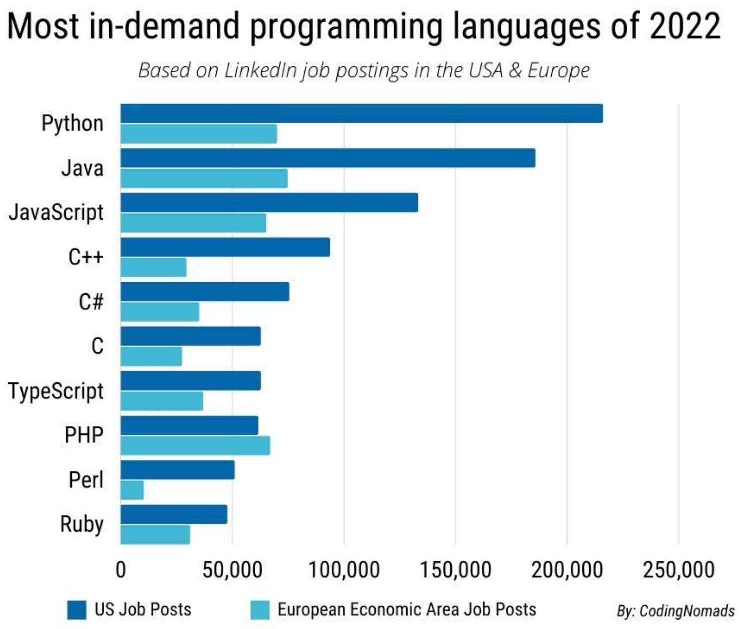


Figure 3.1: Comparing python to other languages.

3.3.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products [5]. For more information about the OpenCV library, you can review (section 2.2) in this work.

3.3.3 PyCharm

PyCharm is the most popular IDE (Integrated Development Environment) for Python scripting language, and includes great features such as excellent code completion and inspection with advanced debugger and support for web programming and various frameworks such as Django and Flask. PyCharm is created by Czech company, Jet brains which focusses on creating integrated development environment for various web development languages like JavaScript and PHP [52].

3.3.3.1 PyCharm features

Besides, a developer will find PyCharm comfortable to work with because of the features mentioned below:

- **Code Completion:**

  PyCharm enables smoother code completion whether it is for built in or for an external package.
- **SQLAlchemy as Debugger**

  You can set a breakpoint, pause in the debugger and can see the SQL representation of the user expression for SQL Language code.
- **Git Visualization in Editor**

  When coding in Python, queries are normal for a developer. You can check the last commit easily in PyCharm as it has the blue sections that can define the difference between the last commit and the current one.
- **Code Coverage in Editor**

  You can run **.py** files outside PyCharm Editor as well marking it as code coverage details elsewhere in the project tree, in the summary section etc.
- **Package Management**

  All the installed packages are displayed with proper visual representation. This includes list of installed packages and the ability to search and add new packages.
- **Local History**

  Local History is always keeping track of the changes in a way that complements like Git. Local history in PyCharm gives complete details of what is needed to rollback and what is to be added.
- **Refactoring**

  Refactoring is the process of renaming one or more files at a time and PyCharm includes various shortcuts for a smooth refactoring process.
- **User Interface of PyCharm Editor**

  The user interface of PyCharm editor includes various features to create a new project or import from an existing project.

3.3.4 The installation of the software and the packages

- **Python**

As we see above Python is a widely used high-level programming language.so how you can install it on your machine.
Unlike most Linux distributions, Windows does not come with the Python programming language by default. However, you can install Python on your Windows server or local machine in just a few easy steps.

Open your web browser and navigate to the downloads for Windows section of the Python.org website.

- **OpenCV**

In this section, I am going to show How to Install OpenCV on Windows with Python. As we see above OpenCV is Open Source and free.

To install OpenCV Via PIP give the following command:

```
pip install opencv-python
```

Pip is a package management system used to install and manage software packages written in Python. You can also install ***opencv-contrib-python*** package to avoid issues during the tracker initialization via the following command:

```
pip install opencv-contrib-python
```

When installing OpenCV you will observe one more thing which is NumPy packages are automatically installed with the OpenCV Python package.

To Test OpenCV Installation use the following command in python: **import cv2**

And if you want to check the version use the following command in python: **cv2.__version__**

- **PyCharm**

This Python IDE from JetBrains has a built-in editor, you have the option of getting the professional edition for free if you are a university student by simply register with your university email address or you can use the free Community Edition.

You can download and install the latest version (*2022.1.1) as of May 2022* from the PyCharm download page, for your operating system.

❖ **Configure PyCharm with Python3**

After you have the software installed in the default location, you must configure PyCharm to work with Python 3. Follow these steps:

1. Launch PyCharm from the program you downloaded/installed.
2. On the welcome screen, go to ***Create New Project***. Choose The correct ***interpreter*** that you will be using for your project (e.g. python3.6). Enter the full path under ***Location*** and select the ***Create*** button in the lower right.
3. Now we will create a new file. Right click on the ***OpenCV*** folder in the project view and select ***New -> Python File***. In the dialog window under *Name*, enter ***object_tracking.py***. This will open the new file in an editor to the right. Now you can enter your code.

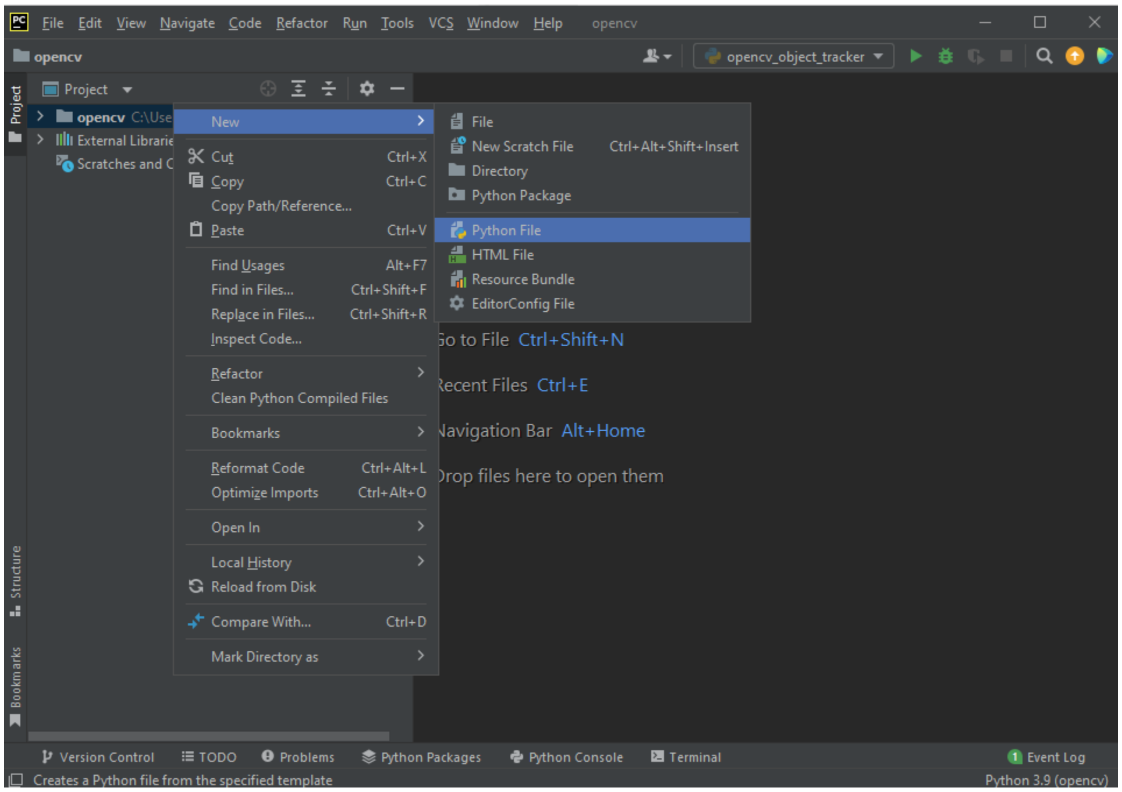


Figure 3.2: Create a Python file in PyCharm.

3.4 Algorithm structure

In order to implement single object tracking in OpenCV, the steps of the algorithm are shown in (Figure 3.3):

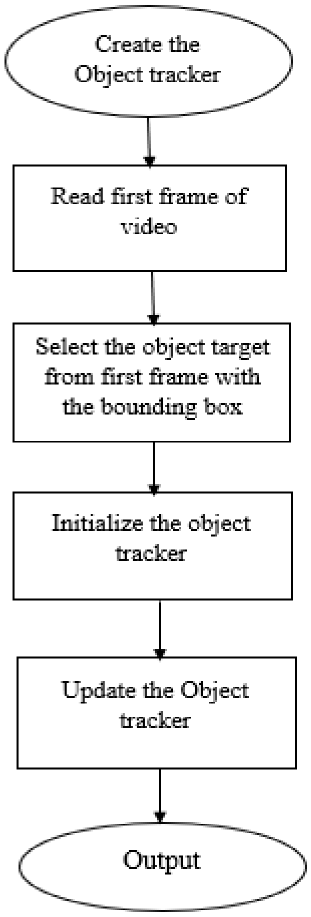

Create the Object tracker
Read first frame of video
Select the object target from first frame with the bounding box
Initialize the object tracker
Update the Object tracker
Output

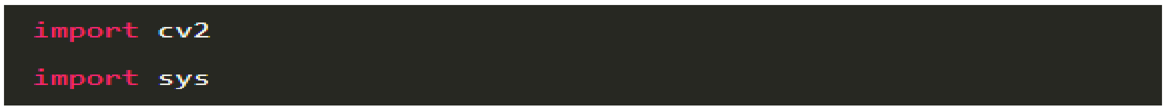
Figure 3.3: Block diagram of object tracking algorithm in OpenCV

3.5 Single object tracking in video by OpenCV with MOSSE, KCF, CSRT

As mentioned in Chapter 2 (see Table 2.1). The OpenCV library provides 8 different ways to track objects. In this part, we are going to implement tracking algorithms based on a correlation filter in OpenCV (MOSSE, KCF, and CSRT algorithms).

First of all, make sure that your environment is ready to work by ensuring that you have OpenCV installed (we recommend OpenCV 3.4+).

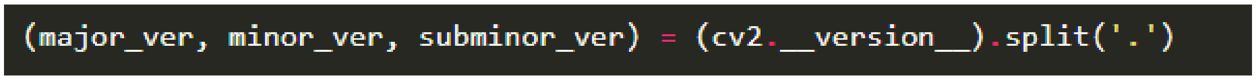
We begin by importing our required packages:



```
import cv2
import sys
```

Figure 3.4: Import libraries.

Now that our packages are imported, we are going to check the OpenCV version:



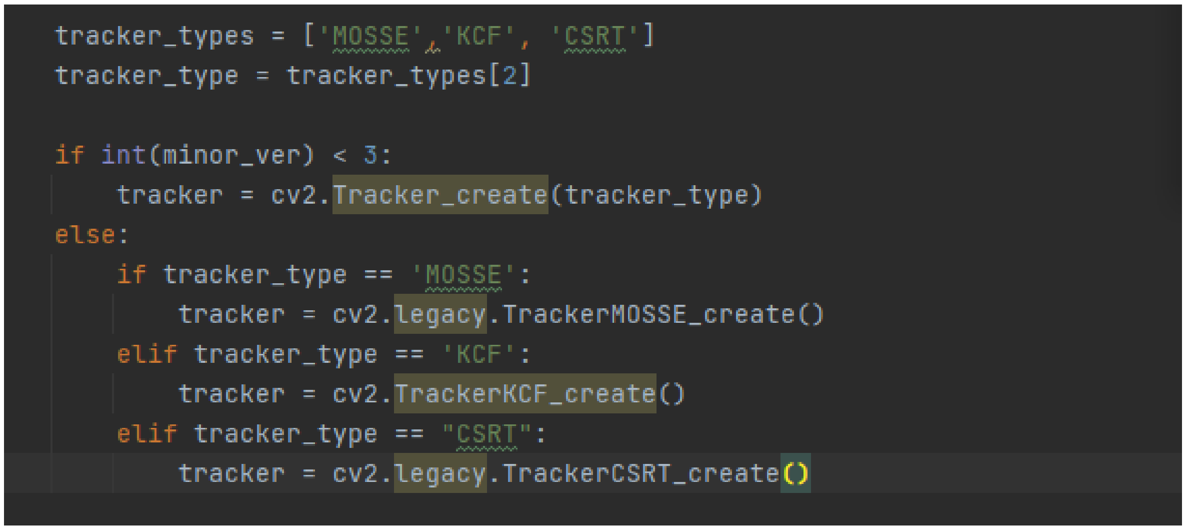
```
(major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
```

Figure 3.5: Checking the OpenCV version.

The **cv2.__version__** function returns the version numbers of the OpenCV library installed in your environment. This check is necessary to do before creating the tracker object. This is because any version of OpenCV lower than 3 has a different module to create a specific type of tracker.

3.5.1 Create the Object tracker

We first save the name of the three trackers that we will implement in this project in a list. Then we check for the version of OpenCV we are working on and then create the tracker object based on the version number (see figure 3.6).



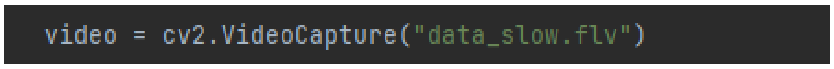
```
tracker_types = ['MOSSE','KCF', 'CSRT']
tracker_type = tracker_types[2]

if int(minor_ver) < 3:
    tracker = cv2.Tracker_create(tracker_type)
else:
    if tracker_type == 'MOSSE':
        tracker = cv2.legacy.TrackerMOSSE_create()
    elif tracker_type == 'KCF':
        tracker = cv2.TrackerKCF_create()
    elif tracker_type == "CSRT":
        tracker = cv2.legacy.TrackerCSRT_create()
```

Figure 3. 6: Create the object tracker.

For OpenCV 3.3+, each tracker can be created with its own respective function call such as: **cv2.TrackerKCF_create ()**. Note, that several tracking algorithms have been removed from the official OpenCV release and moved to the “legacy” section. Therefore, the word “legacy” must be added to some trackers when they are created in order for the tracker to function.

3.5.2 Reading the first frame of the video

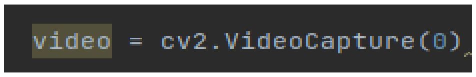
After Setting up the trackers, The VideoCapture class can be used to capture a video file from either the webcam integrated with our machine or a video file saved in our local device. We give the path to our video in the argument of VideoCapture with the following command.



```
video = cv2.VideoCapture("data_slow.flv")
```

Figure 3.7: The video path command.

If you want to use the webcam for tracking, comment on the pervious line of the video path command, and write the following command.

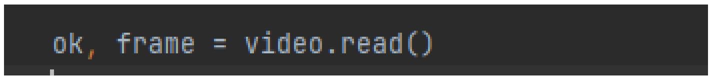


```
video = cv2.VideoCapture(0)
```

Figure 3.8: The command for using the webcam

We further do a couple of checks to see if the video file is properly working or not.

And then we used the '**read()**' function to read the first frame of the video (see figure 3.9).

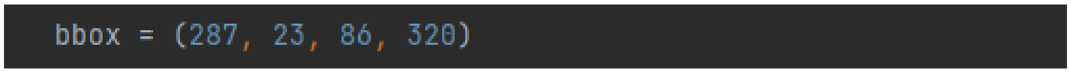


```
ok, frame = video.read()
```

Figure 3. 9: Reading the first frame of the video

3.5.3 Select an object in the first frame with the bounding box

In this part, we first define an initial bounding box in the video (see Figure 3.10), Knowing that the bounding box's coordinates indicate (X, Y, width, height).



```
bbox = (287, 23, 86, 320)
```

Figure 3.10: Define an initial bounding box.

Or we can select a bounding box of our own choice by using **cv2.selectROI** function you will see the window for manual selection (see figure 3.11). This bounding box will contain the object we want to track.

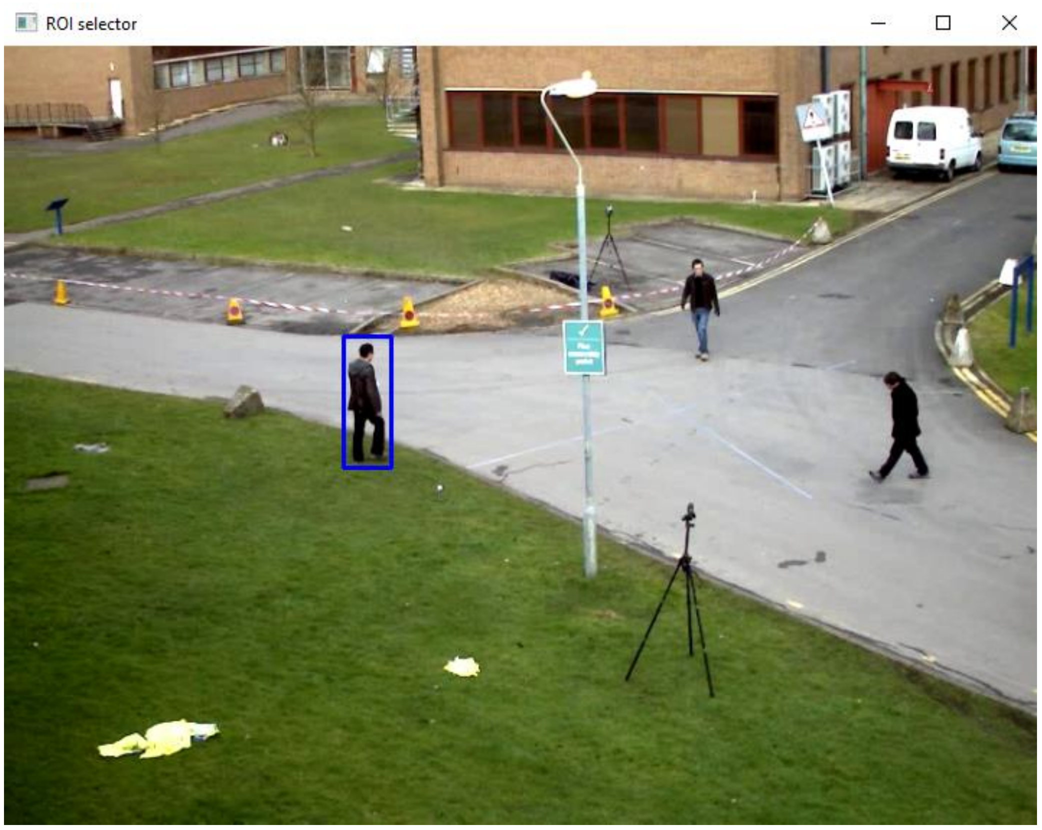


Figure 3. 11: Select an object in the first frame with the bounding box.

3.5.4 Initialize the object tracker

Now, we will initialize the tracker with first frame and bounding box.

```
ok = tracker.init(frame, bbox)
```

Figure 3.12: Initialize the object tracker

3.5.5 Updating the Object tracker and see the output

In order to update the location of the object, we call the **.update()** function to update our frame.

```
ok, bbox = tracker.update(frame)
```

Figure 3.13: Update the object tracker

the tracker "update" function provides us with the up-to-date bounding box. It returns two variables, first is a flag parameter that informs if the tracking process was successful or not and the second returns the position of the tracked object in the frame if and only if the first parameter was true.

Next, in the loop, we start by reading each frame of the video being played. We start the timer with the **cv2.gerTickCount()** function and use the tracker to estimate the trajectory of the object in the video. We use the tracker's estimated trajectory to draw the bounding box around the object of interest. The program continues forever and waits for the 'q' key to be pressed; as soon as the 'q' key is pressed, the while loop breaks, and the tracking stops. We used the OpenCV **cv2.destroy-AllWindows()** function to close all lingering windows if there are any.

The output of tracking with the CSRT tracker is shown in (figure 3.14).

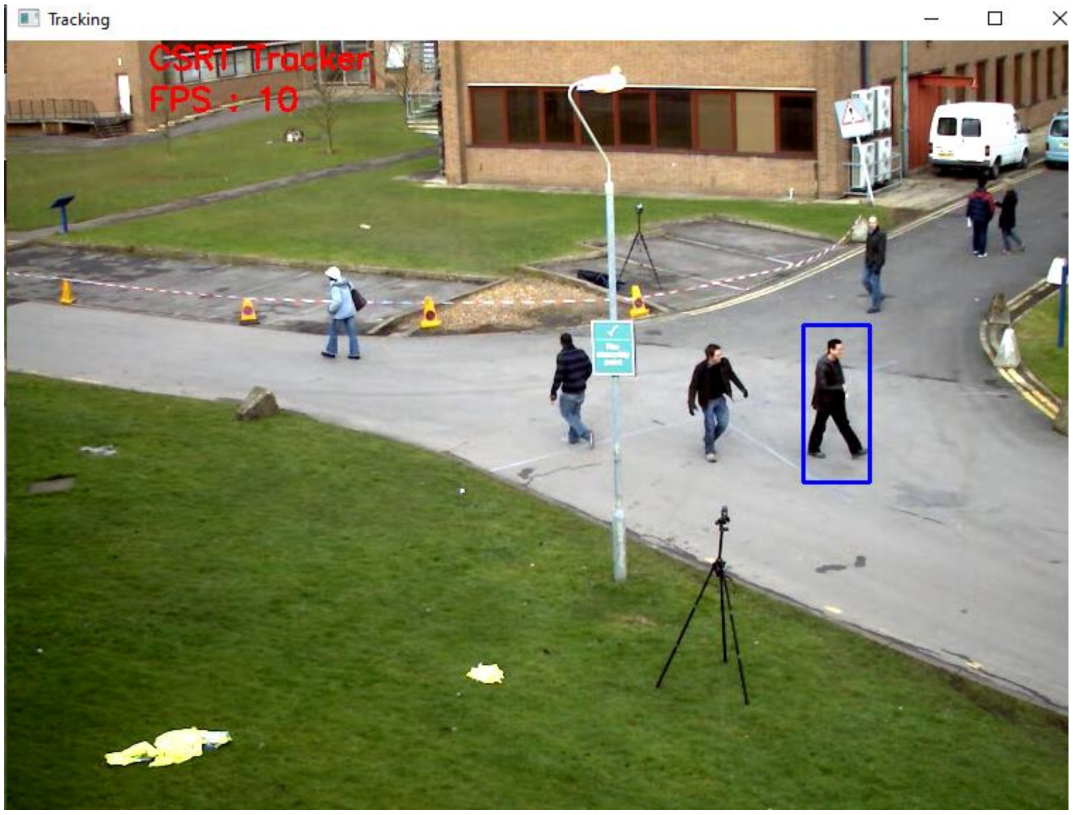


Figure 3.14: The CSRT tracker output

3.6 Performance measurement of a tracking system

The performance evaluation of tracking systems requires a comparison of the results of the algorithms with "optimal" ground truth results. The main performance metrics in object tracking are [18] [71]: Center location error and overlap ratio.

3.6.1 Center location error (CLE)

The center location error (Figure 3.15(a)) is a common measure of measuring the average distance between the centers of the predicted boxes $\{p_i\}_{i=1}^{M}$ and the ground truth $\{g_i\}_{i=1}^{M}$:

$$CLE = \frac{1}{M}\sum_{i=1}^{M} \|p_i - g_i\|$$

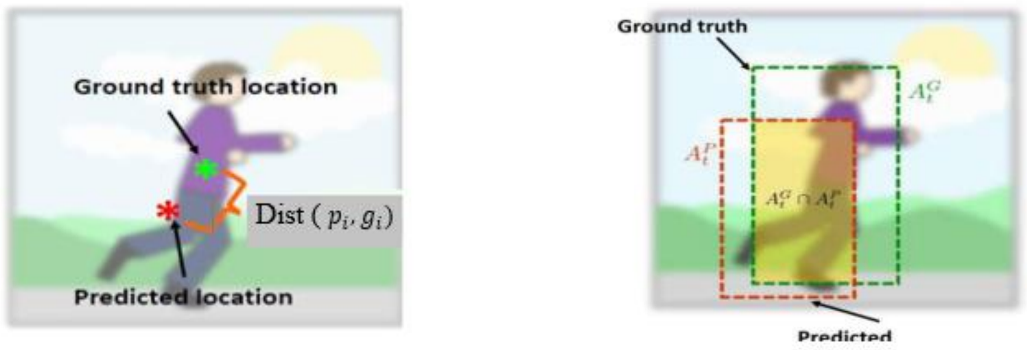
This measure does not account for the size accuracy of the predicted boxes. The usual threshold for comparing accuracy between different trackers is 20 pixels.

3.6.2 Overlap ratio (VOR)

The overlap rate (Figure 3.15(b)) between the predicted box B and the ground truth B' is defined as the ratio of the areas of intersection and union of the boxes:

$$\mathrm{VOR}(B, B') = \frac{|B \cap B'|}{|B \cup B'|}$$

VOR, is a more accurate error measure than center location error since it considers the size of the boxes. The average overlap rate then consists in averaging VOR over all the images in the data-base. We count the number of successful frames whose Overlap rate (VOR) is above the given threshold τ (The threshold is often used for tracking performance evaluation is 0.5).



(a) Center location error

(b) Overlap rate of predicted (red) and ground truth (green) boxes

Figure 3.15: Performance metrics: (a) Center location error, (b) Overlap rate.

3.7 Experimental results and discussions

In this work, we used four image sequences to present the results of OpenCV single object tracking algorithms based on correlation filter: MOSSE, KCF, and CSRT. These sequences contain several tracking challenges. A rectangle represents the target object in the image sequences. The target is selected in the first frame by initializing the bounding box coordinates of the object we will track based on the results obtained in the field. The experimental results are expressed on two levels, quantitative (Center location error and Overlap rate) and qualitative (visually).

3.7.1 Datasets

The image sequences used to evaluate the performance of the OpenCV single object tracking algorithms based on the correlation filter exist in the OTB datasets which are available at: http://cvlab.hanyang.ac.kr/tracker_benchmark/benchmark_v10.html. Each of the video sequences is associated with a ground truth that captures the true interpretation of the scene in terms of objects to follow and is linked to a TXT file describing the position of the target in each image (X, Y, width, height). Table (3.1) illustrates the image sequences used:

Table 3.2:The image sequences used

Image sequence	1st image	Number of frames	Image size	Challenges
Car4		659	360x240	IV, SV
Skiing_ce		1511	1280x720	SV, MB, FM, IPR, OPR
Trellis		569	320x240	IV, SV, IPR, OPR, BC
CarScale		252	640x272	SV, OCC, FM, IPR, OPR

- **OCC**: Occlusion - **MB**: Motion Blur - **OPR**: Out-of-Plane Rotation
- **IV**: Illumination Variation - **FM**: Fast Motion - **BC**: Background Clutters
- **SV**: Scale Variation - **IPR**: In-Plane Rotation.

3.7.2 Comparison between MOSSE and KCF and CSRT trackers in OpenCV

3.7.2.1 Quantitative results

➢ **The comparison between the Center Location Error averages and the Overlap rate for the MOSSE, KCF, and CSRT:**

The table below summarizes the comparison between the three MOSSE, KCF and CSRT trackers, using the CLE and VOR metrics for each sequence tested. We can clearly see that the minimum center localization error and the maximum overlap rate in all sequences tested are obtained by CSRT (12.9131/ 0.577675), especially in the Skiing_ce sequence (15.4868/ 0.7128). This means that CSRT has achieved better performance compared to other trackers. With these results, we can say that the CSRT tracker can track the target object more robustly and more accurately than the MOSSE and KCF.

Table 3.3: The averages of the results obtained for the center localization error and the overlap rate.

	Center location error			**Overlap rate**		
Sequences	**MOSSE**	**KFC**	**CSRT**	**MOSSE**	**KFC**	**CSRT**
Car4	**4.2572**	178.1341	6.3692	0.4736	0.2125	**0.4770**
Skiing_ce	52.9608	39.8926	**15.4868**	0.4367	0.6295	**0.7128**
Trellis	5.7726	127.8408	**3.0546**	0.5587	0.3133	**0.6236**
CarScale	111.7034	115.1554	**26.7418**	0.4172	0.4009	**0.4973**
Averages	43.6735	115.255725	**12.9131**	0.47155	0.38905	**0.577675**

3.7.2.2 Qualitative results

- **Car4 Sequence:**

In the Car 4 sequence, the object to track is a car. And the challenges present in this sequence are Illumination Variation and Scale Variation. Figure (3.16) shows the tracking results by MOSSE, KCF, and CSRT for frames 66, 190, 230, and 488. We can see that, the three trackers can follow the object with precision in the first frames (frame 66), in frame 190, the KCF tracker lost the object due to the illumination in the target area changing significantly. The KCF tracker box reappears in frame 230, but its accuracy has decreased due to the presence of another similar object, while MOSSE and CSRT continue to track the object accurately and robustly. In frame 488, MOSSE and CSRT trackers continued tracking the car until the sequence ended, while the KCF tracker lost the object again.

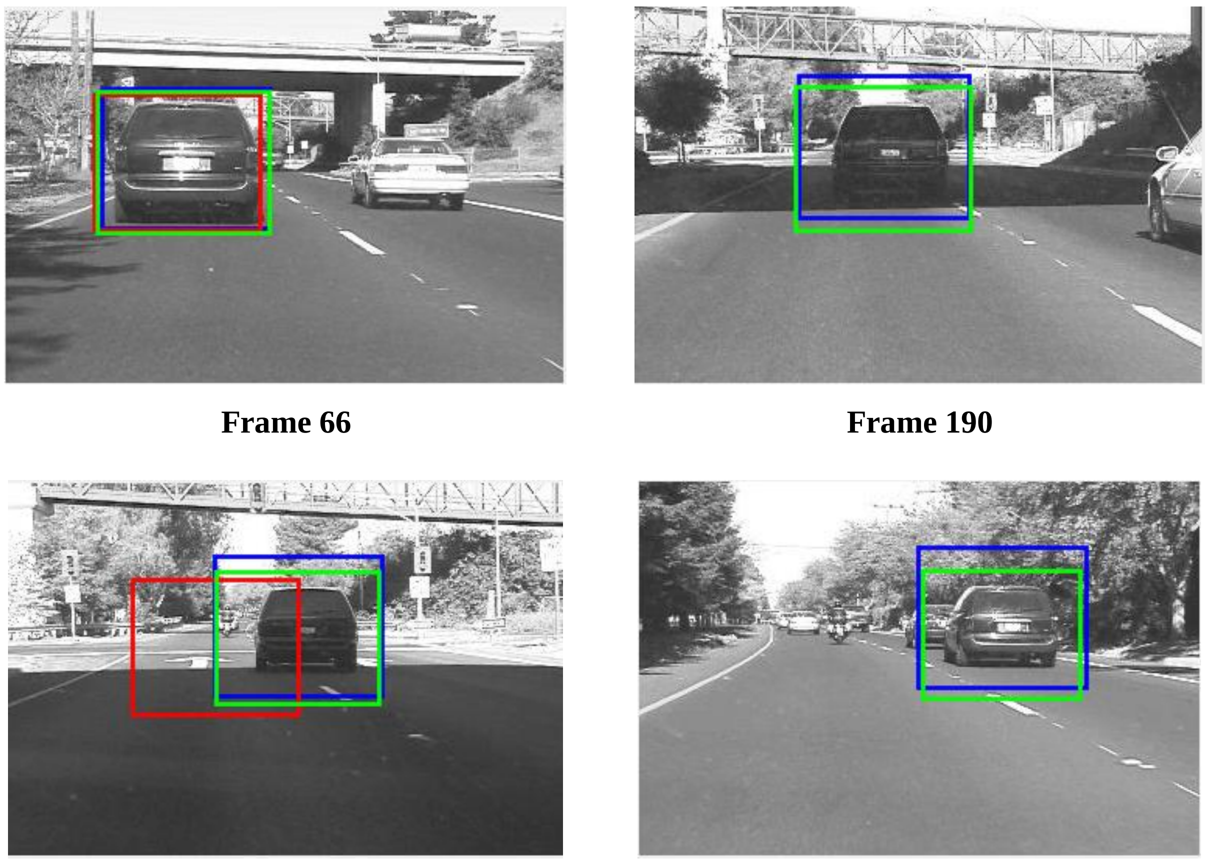
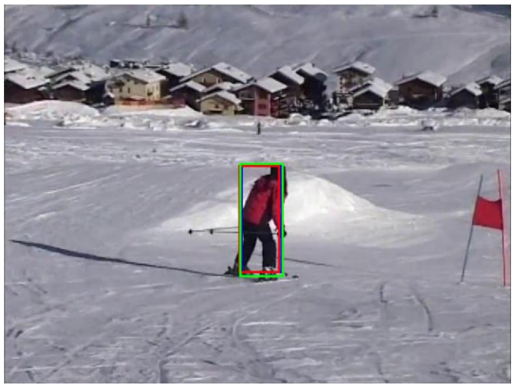
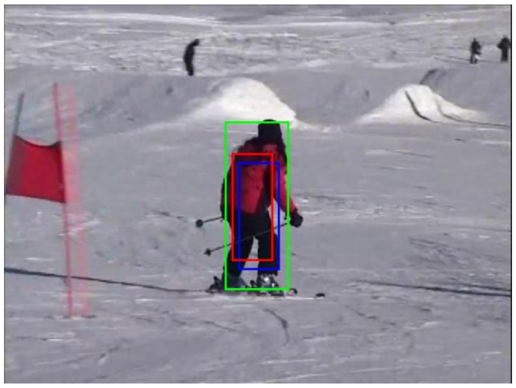


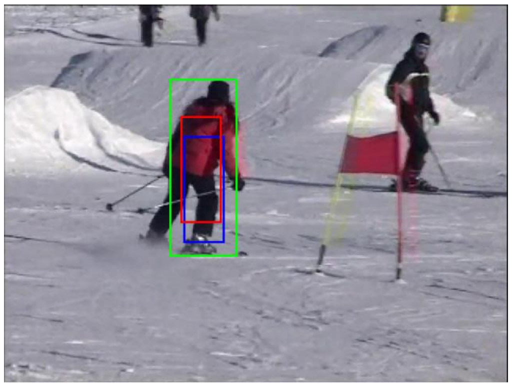
Figure 3.16: Tracking results on the Car4 sequence with MOSSE (blue rectangle), KCF (red rectangle), and CSRT (green rectangle).

- **Skiing_ce Sequence:**

The Skiing_ce sequence represents a girl skiing and the challenges in this sequence are SV, MB, FM, IPR, and OPR. Figure (3.17) shows the tracking results by MOSSE, KCF, and CSRT for frames 23, 64, 224, and 471. At the beginning of the test (frame 23), the three trackers tracked the object with high precision. In the other frames, we can see that the 3 trackers can track the object throughout the sequence, but CSRT is the most accurate and robust, especially in frames 64 and 224 when there is significant scale variation and fast motion.


Frame 23


Frame 64


Frame 224

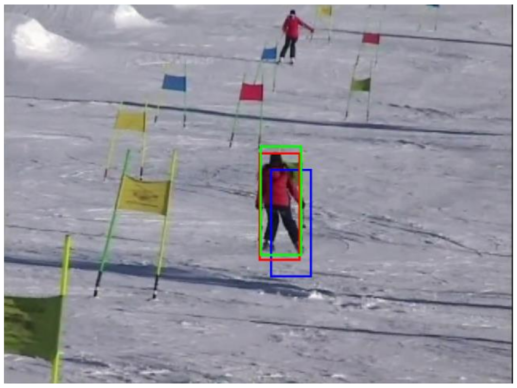

Frame 471

Figure 3.17: Tracking results on the Skiing_ce sequence with MOSSE (blue rectangle), KCF (red rectangle), and CSRT (green rectangle).

- **Trellis Sequence:**

In the Trellis sequence, the object we are going to track is a man's face. and the challenges in this sequence are IV, SV, IPR, OPR, BC. Figure (3.18) shows the tracking results by MOSSE, KCF, and CSRT for frames 5, 95, 285, and 515. We can see that the three trackers can track the object with precision in the first frames (frame 68). However, because of the variations in scale and illumination in frame 95, accuracy decreased. In frame 285 of the sequence, the KCF tracker completely loses the object due to Out-of-Plane Rotation. The two trackers MOSSE and CSRT continue to track the object in frame 515, however the CSRT was more accurate to the target “face” and robust against sequence problems.

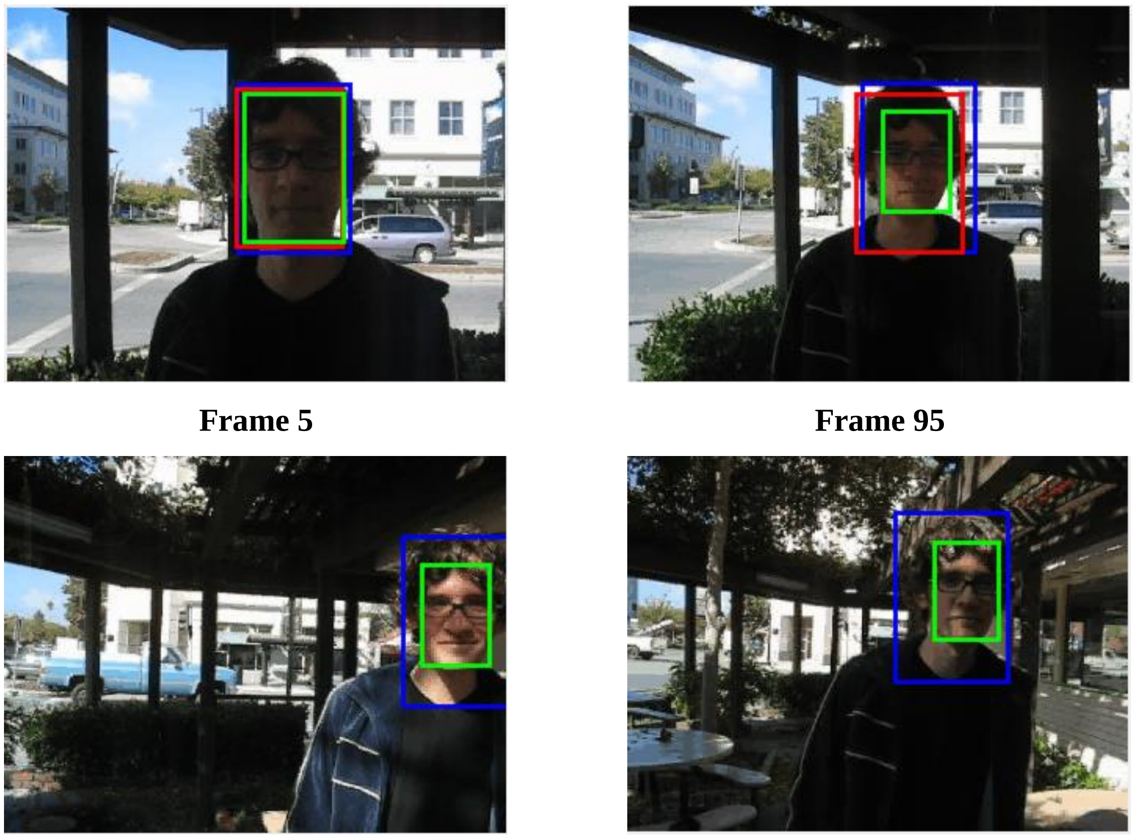


Figure 3.18: Tracking results on the Trellis sequence with MOSSE (blue rectangle), KCF (red rectangle), and CSRT (green rectangle).

- **CarScale Sequence:**

The thing we are going to track in CarScale sequence is a car. And the challenges present in this sequence are SV, OCC, FM, IPR, OPR. Figure (3.19) shows the tracking results by MOSSE, KCF, and CSRT for frames 26, 150, 165, and 235. Despite the challenges it faced that affected its tracking accuracy as indicated in the shown frames, CSRT was able to track the target object to the end of the sequence. In contrast to MOSSE and KCF, they completely lost the object between frames 150 and 165 due to the occlusion of the tree and our target. In frame 235, we can see that the CSRT continued to track the car until the sequence ended but with low precision because of the scale variation and the fast motion in these frames.

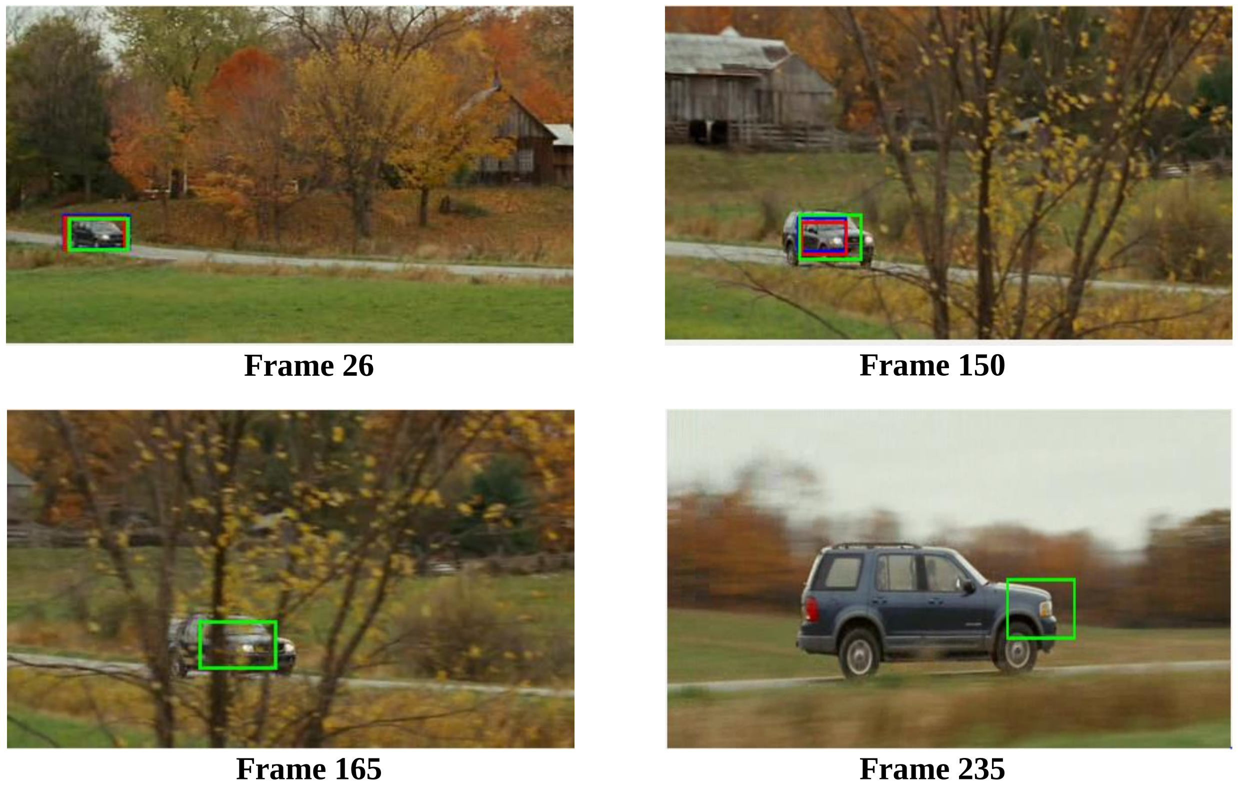


Figure 3.19: Tracking results on the CarScale sequence with MOSSE (blue rectangle), KCF (red rectangle), and CSRT (green rectangle).

3.8 Conclusion

In this chapter, we implemented and compared MOSSE, KCF, CSRT algorithms for single object tracking in OpenCV library. We saw first the hardware and the software that are used in this project along with the installations, then we viewed the steps to tracking a single object in video and webcam with OpenCV. Then we started to test our algorithms on image sequences, we find different challenges that oppose object tracking (Occlusion, scale and Illumination Variation, Background Clutters, fast motion, and motion blur...). Then we presented the results obtained for each sequence for the MOSSE, KCF and CSRT algorithms and we compared the three trackers by the two metrics: center location error and overlap rate. The obtained results showed that CSRT tracker is more accurate and more robust in dealing with various of difficulties than MOSSE and KCF in single object tracking by OpenCV.

General Con-
clusion

General conclusion

One of the main goals of computer vision is to enable computers to replicate the basic functions of human vision such as motion perception and scene understanding. To achieve the goal of intelligent motion perception, much effort has been spent on visual object tracking, which is one of the most important and challenging research topics in computer vision. Essentially, the core of visual object tracking is to robustly estimate the motion state (i.e., location, orientation, size, etc.) of a target object in each frame of an input image sequence [20].

Despite extensive studies during the past several decades, tracking objects under unconstrained scenarios is still a complex and difficult task due to many practical challenges, including illumination change, occlusion, deformable objects, noise corruption, viewpoint variations, and motion blur. Furthermore, the proliferation of video data and new data-acquisition devices has stimulated a great deal of interest in building more intelligent tracking algorithms [21].

The OpenCV library provides 8 different object tracking methods using online learning classifiers [5], [33]. The term online refers to algorithms that are trained using very few examples at run time, in contrast to an offline classifier, which may need thousands of examples to train. Most modern solutions to the object tracking problem assume the presence of a pre-trained classifier (an offline classifier), that allows us to accurately determine the object we track, whether it is a car, person, animal, etc. But, what if the user can’t find a suitable classifier or train their own? In this case, OpenCV offers a number of pre-built algorithms developed explicitly for the purpose of object tracking[33], [34].

The aim of our work was to track a single object by pre-implemented tracking algorithms available in the OpenCV library. From the eight algorithms in the OpenCV library, we focused on the MOSSE, KCF, and CSRT algorithms, which are based on the correlation filter technique. We chose these algorithms to implement based on our research because they are the most recommended tracking algorithms in OpenCV; we tested them and compared their results.

In this thesis, we have implemented the MOSSE, KCF, and CSRT algorithms on OTB database image sequences which contain several challenges (Occlusion, scale Variation, Background Clutters, fast motion, Illumination Variation, and motion blur...), and we made a comparative study between the three methods from a qualitative point of view (visual) and from a quantitative (center location error and overlap rate).

The results obtained show that the CSRT tracker is more accurate and more robust than MOSSE and KCF in dealing with various of difficulties in single object tracking by OpenCV.

In perspective, we propose the possibility of extending this work by further experimenting with the different object tracking algorithms in OpenCV to find effective solutions for different use cases.

Bibliography

Bibliography

[1] D. Dzigal, A. Akagic, E. Buza, A. Brdjanin, and N. Dardagan, “Forest Fire Detection based on Color Spaces Combination,” in *ELECO 2019 - 11th International Conference on Electrical and Electronics Engineering*, Nov. 2019, pp. 595–599. doi: 10.23919/ELECO47770.2019.8990608.

[2] J. J. Titano *et al.*, “Automated deep-neural-network surveillance of cranial images for acute neurologic events,” *Nature Medicine*, vol. 24, no. 9, pp. 1337–1341, Sep. 2018, doi: 10.1038/s41591-018-0147-y.

[3] S. Stabinger, A. Rodríguez-Sánchez, and J. Piater, “25 years of CNNS: Can we compare to human abstraction capabilities?,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9887 LNCS, pp. 380–387. doi: 10.1007/978-3-319-44781-0_45.

[4] Y. Wu, J. Lim, and M. H. Yang, “Object tracking benchmark,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834–1848, Sep. 2015, doi: 10.1109/TPAMI.2014.2388226.

[5] A. Brdjanin, N. Dardagan, D. Dzigal, and A. Akagic, “Single object trackers in opencv: A benchmark,” in 2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA). IEEE, 2020, pp. 1–6.

[6] P. Janku, K. Koplik, T. Dulik, and I. Szabo, “Comparison of tracking algorithms implemented in OpenCV”, doi: 10.1051/04.

[7] A. Yilmaz, O. Javed, and M. Shah, “Object tracking: A survey,” *ACM Computing Surveys*, vol. 38, no. 4. Dec. 25, 2006. doi: 10.1145/1177352.1177355.

[8] Y. Wu, J. Lim, and M. H. Yang, “Online object tracking: A benchmark,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2411–2418. doi: 10.1109/CVPR.2013.312.

[9] S. Liu, D. Liu, G. Srivastava, D. Połap, and M. Woźniak, “Overview and methods of correlation filter algorithms in object tracking,” *Complex & Intelligent Systems*, vol. 7, no. 4, pp. 1895–1917, Aug. 2021, doi: 10.1007/s40747-020-00161-4.

[10] S. Medouakh. “ Détection et suivi d’objets.’’Thèse de doctorat. Université de Biskra, 2019.

[11] I. LAMARI. “ Suivi d’objets avec Mean shift en utilisant BWH et CBWH.’’Thèse de Master, université de Mohamed Khider Biskra.

[12] W, Bouchir. “Suivi d’objet par caractéristique locales encadrant la structure ’’Thèse de Doctorat, Université Montréal, 2014.

[13] N. Dardagan, A. Brđanin, D. Džigal, and A. Akagic, “Multiple Object Trackers in OpenCV: A Benchmark,” Oct. 2021, [Online]. Available: http://arxiv.org/abs/2110.05102

[14] Brulin, Mathieu. “Analyse sémantique d'un trafic routier dans un contexte de vidéo surveillance.’’Diss. Bordeaux 1, 2012.

[15] S. Avidan, “Support vector tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence,’’ PAMI, 26(8):1064-1072 (2004).

[16] F. Porikli, A. Yilmaz, F. Porikli, and A. Yilmaz, “Object Detection & Tracking,” 2012. [Online]. Available: http://www.merl.com

[17] Mirmehdi, M., Xie, X., Suri, J. (eds.): Handbook of Texture Analysis. Imperial College Press (2008).

[18] D. Sun, S. Roth, and M. J. Black, “Secrets of optical flow estimation and their principles,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2432–2439. doi: 10.1109/CVPR.2010.5539939.

[19] S. He, Q. Yang, R. W. H. Lau, J. Wang, and M. H. Yang, “Visual tracking via locality sensitive histograms,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2427–2434. doi: 10.1109/CVPR.2013.314.

[20] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, and A. van den Hengel, “A Survey of Appearance Models in Visual Object Tracking,” Mar. 2013, [Online]. Available: http://arxiv.org/abs/1303.4803

[21] Yang. H. : Vers un suivi robuste d’objets visuels : sélection de propositions et traitement des occlusions. (2016).

[22] D. Riahi, Suivi multi-objets par la détection : « Application à la vidéo surveillance. Diss ». Ecole Polytechnique, Montréal (Canada), 2016.

[23] X. Mei, H. Ling: « Robust visual tracking using l1 minimization ». In IEEE 12th International Conference on Computer Vision, pp. 1436–1443 (2009).”

[24] D. A. Ross, J. Lim, R. S. Lin, and M. H. Yang, “Incremental learning for robust visual tracking,” *International Journal of Computer Vision*, vol. 77, no. 1–3, pp. 125–141, May 2008, doi: 10.1007/s11263-007-0075-7.

[25] H. Grabner, M. Grabner, and H. Bischof, “Real-time tracking via on-line boosting,” in *BMVC 2006 - Proceedings of the British Machine Vision Conference 2006*, 2006, pp. 47–56. doi: 10.5244/c.20.6.

[26] B. Babenko, M.-H. Yang, and S. Belongie, “Visual tracking with online multiple instance learning,” in 2009 IEEE Conference on computer vision and Pattern Recognition. IEEE, 2009, pp. 983–990.

[27] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection,” IEEE transactions on pattern analysis and machine intelligence, vol. 34, no. 7, pp. 1409–1422, 2011.

[28] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-backward error: Automatic detection of tracking failures," in 2010 20th International Conference on Pattern Recognition. IEEE, 2010, pp. 2756–2759.

[29] B. D. Lucas, T. Kanade, et al. (1981). "An iterative image registration technique with an 1pplication to stereo vision.''In IJCAI, volume 81, pages 674–679

[30] D. Comaniciu, V. Ramesh, and P. Meer, "Real-Time Tracking of Non-Rigid Objects using Mean Shift." In Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on, volume 2, pages 142–149. IEEE.

[31] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE, 2010, pp. 2544–2550.

[32] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," IEEE transactions on pattern analysis and machine intelligence, vol. 37, no. 3, pp. 583–596, 2014.

[33] Satya Mallick, "Object Tracking using OpenCV (C++/Python)," Feb. 17, 2017. https://learnopencv.com/object-tracking-using-opencv-cpp-python/ (accessed Jun. 14, 2022).

[34] brouton lap, "A Complete Review of the OpenCV Object Tracking Algorithms [Blog post]." https://broutonlab.com/blog/opencv-object-tracking (accessed Jun. 14, 2022).

[35] A. Lukezic, T. Vojir, L. Cehovin Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter with channel and spatial reliability," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6309–6318.

[36] D. Held, S. Thrun, and S. Savarese, "Learning to Track at 100 FPS with Deep Regression Networks," in European Conference on Computer Vision. Springer, 2016, pp. 749–765.

[37] S. P. Singh, A. Mittal, M. Gupta, S. Ghosh, and A. Lakhanpal, "Comparing Various Tracking Algorithms In OpenCV," 2021.

[38] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to OpenCV," in 2012 proceedings of the 35th international convention MIPRO. IEEE, 2012, pp. 1725–1730.

[39] OpenCV team. (2020). OpenCV [Blog post]. Retrieved from https://opencv.org/about/

[40] Z. bin Shafi, "Real Time Object detection and Tracking Using Open-CV," International Journal of Scientific Research & Engineering Trends Volume 8, Issue 2, Mar-Apr-2022, ISSN (Online): 2395-566X.

[41] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the Circulant Structure of Tracking-by-detection with Kernels."

[42] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Discriminative Scale Space Tracking," Sep. 2016, [Online]. Available: http://arxiv.org/abs/1609.06141

[43] M. Danelljan, G. Häger, S. Khan, and M. Felsberg, “Accurate Scale Estimation for Robust Visual Tracking Ours ASLA SCM Struck LSHT.” In: British machine vision conference, pp 1–5.

[44] Danelljan M, Robinson A, Khan F S, Felsberg M (2016) Beyond correlation filters: Learning continuous convolution operators for visual tracking. In: European conference on computer vision, pp 472–488

[45] “Learn Object Tracking in OpenCV Python with Code Examples - MLK - Machine Learning Knowledge.” [Online]. Available: https://machinelearningknowledge.ai/learn-object-tracking-in-opencv-python-with-code-examples/ (accessed Jun. 15, 2022).

[46] F. Gong, H. Yue, X. Yuan, W. Gong, and T. Song, “Discriminative correlation filter for long-time tracking,” *Computer Journal*, vol. 63, no. 3, pp. 461–468, Mar. 2020, doi: 10.1093/comjnl/bxz049.

[47] Stack Overflow Documentation. (n.d.). Python® Notes for Professionals [e-book]. Retrieved from https://goalkicker.com/PythonBook/

[48] Guido van Rossum and the Python development team, “Python Tutorial Release 3.6.4,” 2018.

[49] Srinath, K. R. (2017). Python–The Fastest Growing Programming Language. International Research Journal of Engineering and Technology (IRJET), 4(12), 354-357.

[50] Tutorials Point (I) Pvt. Ltd., *pycharm_tutorial [e-book]*. 2018. Accessed: Jun. 15, 2022. [Online]. Available: https://www.tutorialspoint.com/pycharm/pycharm_tutorial.pdf